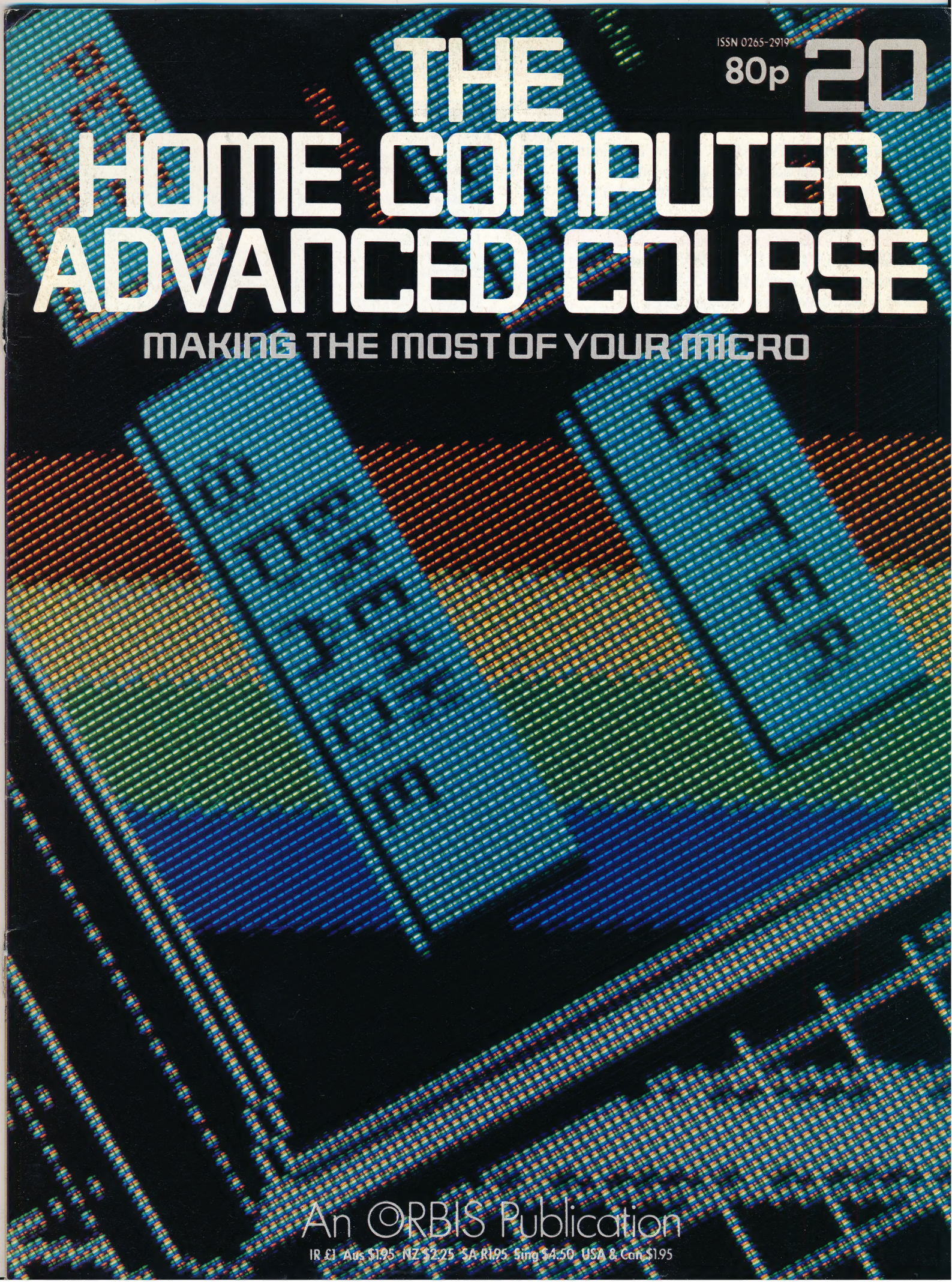
ISSN 0265-2919

80p 20

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

●The Brother EP44 is a low cost computer printer that can also be used as a word processor — thanks to its 15-character display, memory and software — and as a terminal to work with larger computers. We take a look at this sophisticated machine.

●Trying to create graphics from BASIC can be hard work. We test four 'digital tracers', machines that copy the movement of a pen onto the computer screen. These allow maps or other printed graphics to be copied, or can create new graphics.

QUIZ

1) What is the difference between source code and object code?

2) How many pixels can be represented in one byte in mode 5 on the BBC micro, and why?

3) To write a program in BASIC on the Coleco Adam, what must the user do after switching on the machine?

4) What is the name of the company you work for when playing Jet Pac?

Answers To Last Week's Quiz

A1) Bi-directional printing means printing alternate lines right to left and left to right. Therefore, the printer does not waste time in carriage returns.

A2) A BCC instruction checks to see if the carry flag is set to 1 or 0. If it is 1, the program will perform an addition; if not, it will skip the addition and go on to the rotation operations.

A3) To allow the BBC to run a CP/M-style disk operating system.

A4) Gridrunner is marketed by Salamander under licence from Llamasoft.

QUIZ

COVER PHOTOGRAPHY BY IAN McKINNELL

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistants** Liz Dixon, Gary Capps-Jenner; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Researchers** Melanie Davis, Martha Ellen Zenfell; **Contributors** Chris Bidmead, Steve Colwill, Ted Ball, Max Phillips, Matt Nicholson, Dave Watkins, Graham Storrs, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

WRITING FOR THE SCREEN

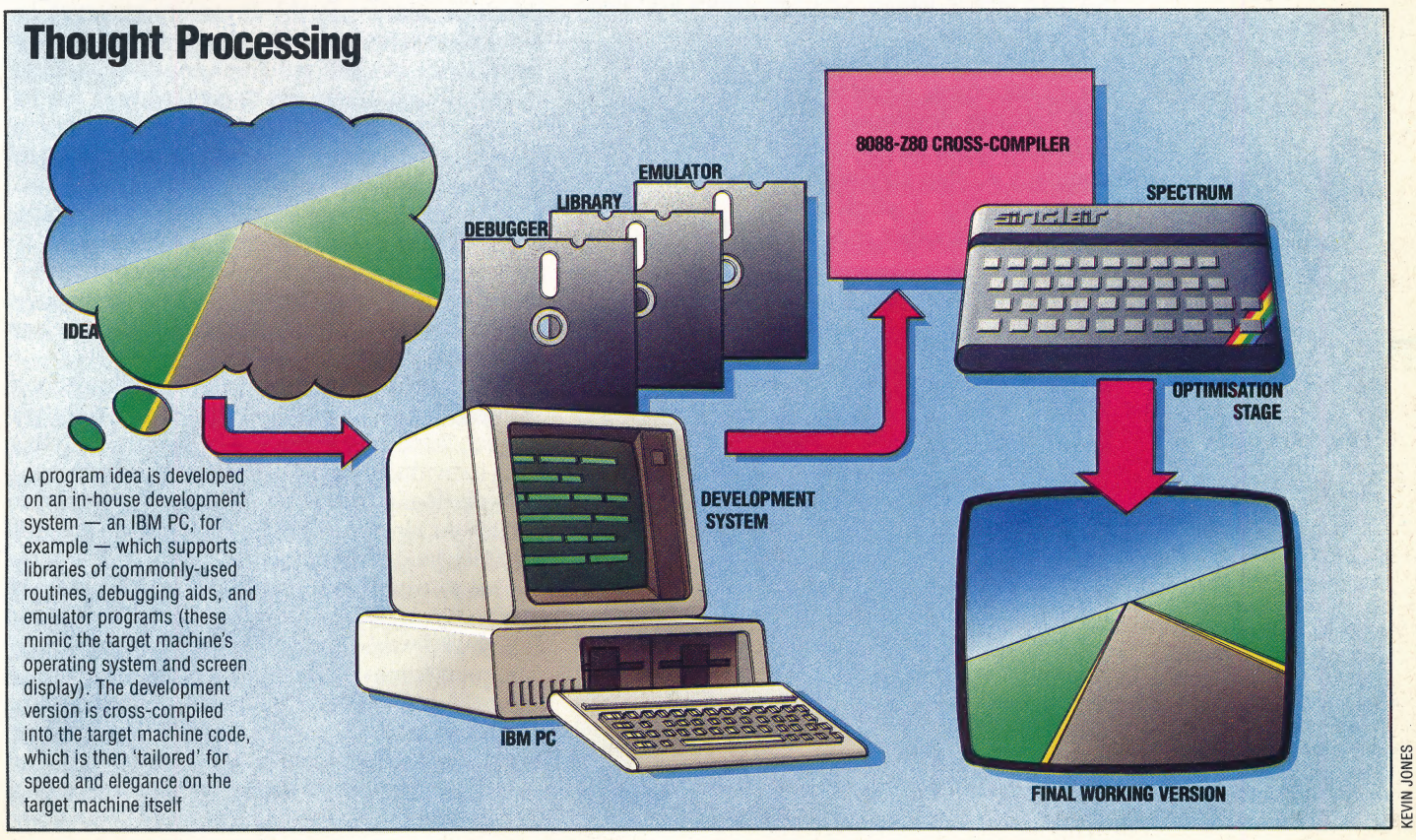
Many home programmers dream of writing a best seller. But they rarely realise what they are up against. Professional software companies have enormous resources behind them to help produce their chart-toppers. One UK software house has over £250,000 worth of minicomputers dedicated to creating packages for home micros.

Expensive equipment doesn't necessarily mean successful programs; some amateur writers have managed to make a small fortune with software they've produced on a Spectrum at home. All the same, home programming whizzkids are becoming an endangered species, especially with the development of the big software houses over the past few years. Their powerful computers and sophisticated programming aids give them a real advantage over the home computer owner and allow their programmers to be more productive.

One of the most important attributes of serious software for home machines is the speed of operation; and this means that programs need to be written (at least in part) in machine code. But machine code is extremely difficult to work with — in particular, machine code programmers need other pieces of software to help write their programs. At the very least, an assembler program will be required to translate the programmer's source code into the object code that the machine understands, and this can be quite a challenging job if a big program is involved. Many software writers work in this way. To write a program for the Spectrum, for example, they will use an assembler program running on the same machine. This method has its limitations.

Primarily, the quality of assembler programs available for home machines is poor. Even the simplest of these packages will use up considerable amounts of memory, and therefore limit the size of the programs that can be written with it. Many home machines are also extremely unpleasant to work with for long periods of time: poor keyboards, poor displays and, in some cases, a lack of disk drives, can make using such equipment a tortuous task.

For these reasons most professional companies don't use the micro that the program is intended for (called the 'target machine'), but use business computers with special software (known as 'development systems') instead. Programmers using these machines often write in languages such as PASCAL and C. They use versions of these



A program idea is developed on an in-house development system — an IBM PC, for example — which supports libraries of commonly-used routines, debugging aids, and emulator programs (these mimic the target machine's operating system and screen display). The development version is cross-compiled into the target machine code, which is then 'tailored' for speed and elegance on the target machine itself

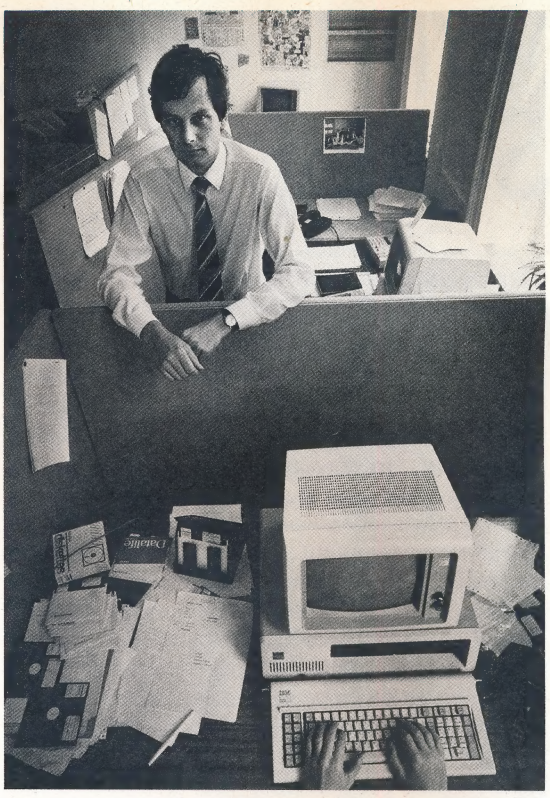
Intelligent Software
Specialising in strategy games such as chess, IS uses IBMs and Apples, with its own specially-developed interfaces, to develop software. Dividing programs into their machine-dependent and universal segments makes it easier for IS to support a range of computers and dedicated chess-playing machines

languages known as cross-compilers or cross-assemblers, which permit the work to be done on a micro that uses an 8086 processor, for example, while the programs that are produced will work on machines with Z80 processors. These cross-compilers are high-level languages (like BASIC), which makes them easy for the programmer to use, but the programs they create are written in machine code. Skilled machine code programmers scrutinise the programs that are developed, and often succeed in further optimising them.

Clearly, a development system has an enormous advantage over the home micro. A disk-based assembler, or one making use of expanded RAM space to store larger tables, will work more efficiently than an assembler that has to be wound in off tape and operates in the confines of a home micro. Debugging routines can be added into the development version of the code, with no worries about the code being too big for memory. It is also far better to work on a business computer that has a good keyboard, sharp display and disk drives.

A firm that makes use of this technique of program development is Intelligent Software (IS), founded in 1981 out of a pooling of experience between David Levy, the chess specialist, and Robert Madge's ANT Microware. The company specialises in strategy games, mostly written on contract for all the popular home micros. They also develop the software side of dedicated chess machines. Although there are no rapid combat displays in games like chess and bridge, a great deal of computation goes on behind the scenes. So, like arcade games, strategy games also need the speed of assembler-written software.

As well as using the target machines themselves for development, IS uses IBM PCs and Apples with specially developed interfaces to allow code to be exchanged across its range of machines. The company is often involved in conversion projects — transferring a chess game, say, from one computer to another — so its programmers have learned to write code in a form that is easily segmented. One level of segmentation that proves useful when the time comes to hand code from one processor to another is the division of the program into playing code and input/output code. The I/O code on the new machine will have different port or memory addresses, and will perhaps be strategically different too (polling replaced by interrupts, and so forth). Ingenuity may well be required to get round the hardware limitations, but there won't be a forbidding quantity of input/output. Playing code, on the other hand, will be there in abundance, but because it is isolated from the hardware (except, of course, the processor) its conversion will be straightforward.

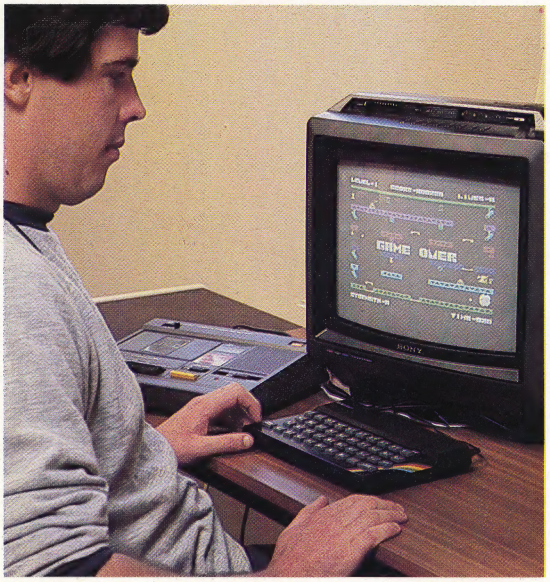

TONY SLEEP

IS wants to avoid restricting programmers' inventiveness, so there are very few 'house rules' to govern the writing of code. One important point that they insist on, though, is that source code includes numerous comments, so it is always clear what the routines are doing.

Where programmers are working at home for a software house, each developing his or her own project, there is little pooling of resources. In this case, individuality is preserved at the cost of a great deal of duplicated effort, because the code for similar routines has to be reinvented by each separate programmer.

One software company, Psion, is making use of computers even larger than the IBM PC. Among British software houses writing for the home computer games market, Psion is unique in doing the bulk of its development on minicomputers. The company's hardware installation alone is worth a quarter of a million pounds.

Psion began as a company by developing software for the ZX81 — and used ZX81s to do it.



BOB BROMIDE

Visions
Programmers working from home on the target machines provide the bulk of Visions' programming effort. After the game concept and scenarios have been decided, the component routines are developed in native Assembly language (Z80 or 6502) using assemblers such as HiSoft DevPak on the Spectrum

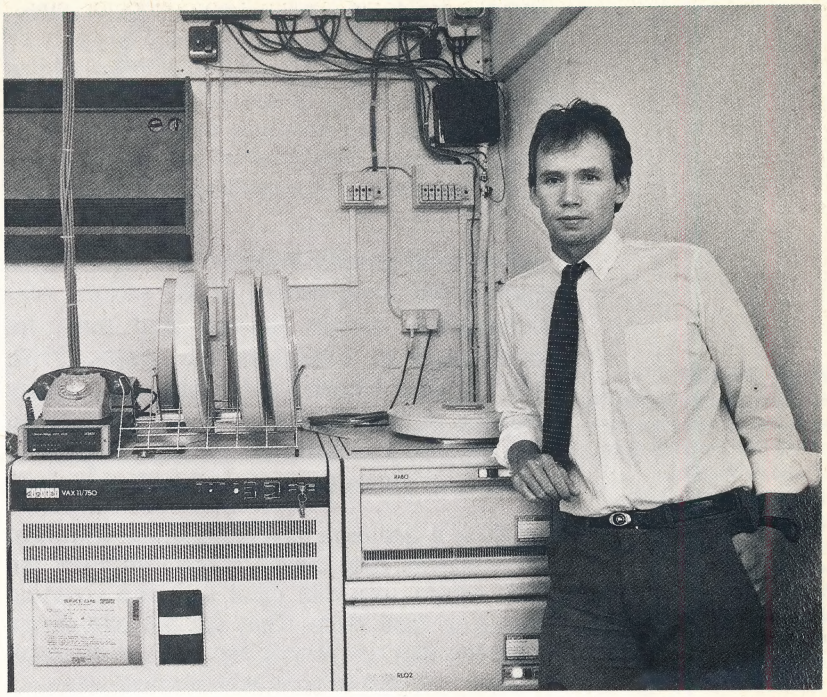
When it went on to create the Horizons tape issued with every Spectrum, Psion bought a TRS-80 with disks, a machine that uses the same Z80 processor, and built a special interface between the two machines. But by August 1982, the company decided that it couldn't go on knocking up a completely different development system every time a new home computer went on the market. So it ploughed back profits into buying heavy-weight hardware with plenty of spare processing power. In principle, this hardware should be flexible enough to cope with whatever computers the future might bring. The machines chosen were a pair of Vax 750s, running the DEC operating system VMS.

The Vax 750s brought two advantages to Psion: the quality of the software provided by DEC, with the opportunity it provides to create specially designed software aids, and the sheer 'muscle-power' of the operating system and hardware combination. There is plenty of room for a collection of software aids like compilers, libraries of common subroutines, and debugging programs, all shared between the 16 to 20 programmers who may be logged on to a single machine at the same time. The two machines allow software to be easily transferred from one to the other when needed.

Libraries of common subroutines had already been part of Psion's philosophy in the TRS-80 days, but on a dual floppy system swapping the data between disks became tedious. The new Vax machines allow teams of writers to work together, sharing common project libraries from which modules can be called up almost instantly, and libraries can even be shared between teams working on different projects. This is the big advantage of a timesharing system — and as an added bonus it will also take care of their administrative work without having to interrupt the programmers. Psion plans to add a third Vax to shoulder the administration tasks, leaving two machines free for software production.

Even if you could afford it, you would be wrong to think that going out and buying a Vax would instantly put you on a par with Psion. Very little of this well-developed work environment has been handed to Psion on a plate by DEC. It has taken a lot of hard slog, both to get simple tasks performed efficiently and reliably, and in the large number of hand-wrought software aids and utilities (written in C) that Psion has added.

Psion uses C, an 'intermediate-level' language that can produce reasonably compact and fast object code for 16-bit chips like the 8086, although this is far from the case for eight-bit C compilers. So in writing for target machines like the Spectrum it has been necessary for Psion to develop its own special techniques. Psion is not keen to disclose its secrets, but it is known the company used C to write its own compiler, which in default of a name gets called 'our table language'. This looks a little like C, is portable between different processors, and creates extremely efficient code.



TONY SLEEP

Psion
In 1982 this company bought a pair of Vax 750 minicomputers as the basis of their software development system. Each machine allows up to 20 programmers at a time to use the range of cross-compilers, software libraries and debuggers for creating and translating programs

There is a universal rule that system maintenance and writing in-house programming aids like the table language generally takes 30 per cent of the entire programming effort, but Psion finds the extra time well worth while. Having the source code developed in-house means total ownership: you can take it to pieces and improve or adapt it in a way completely impossible with commercially acquired software. If a bug turns up in bought-in software it is difficult or impossible to get it fixed, and there is usually no question of making internal changes.

The special software bought by Psion includes programs that are exact simulations of popular microprocessors, such as the Z80 and 6502. Thus, the giant Vax computers can be made to behave just as if they were Commodore 64s or Spectrums. Despite the power of the Vax computers, the simulators run at a fraction of the speed of the target machine. The advantage is that they allow the programmer to look at the contents of every register inside the microprocessor at any stage of the program. This is particularly useful for tracking down bugs in programs. Normally, when a machine code program goes wrong and crashes, the programmer can't tell what went wrong. Psion can thus save many hours of debugging.

Much of Psion's recent development effort has gone into producing the suite of four standard business programs that are provided with the Sinclair QL. The Motorola 68000 family of chips, one of which powers the QL, was designed around high-level languages, and C programs compile down so efficiently on these chips that writing in assembler becomes unnecessary. If all home computers followed the QL lead, C could replace assembler completely, and Psion and the smaller software houses could leave the Dickensian work of hand-coded translation behind forever.

FUN IN A DUNGEON

Adventure games usually pit the player against the imagination of the writer, who determines certain actions that must be performed to complete the game. With MUD, the Multi-User Dungeon, many players, at their home computer terminals, can log into a mainframe machine to play an adventure game against each other.

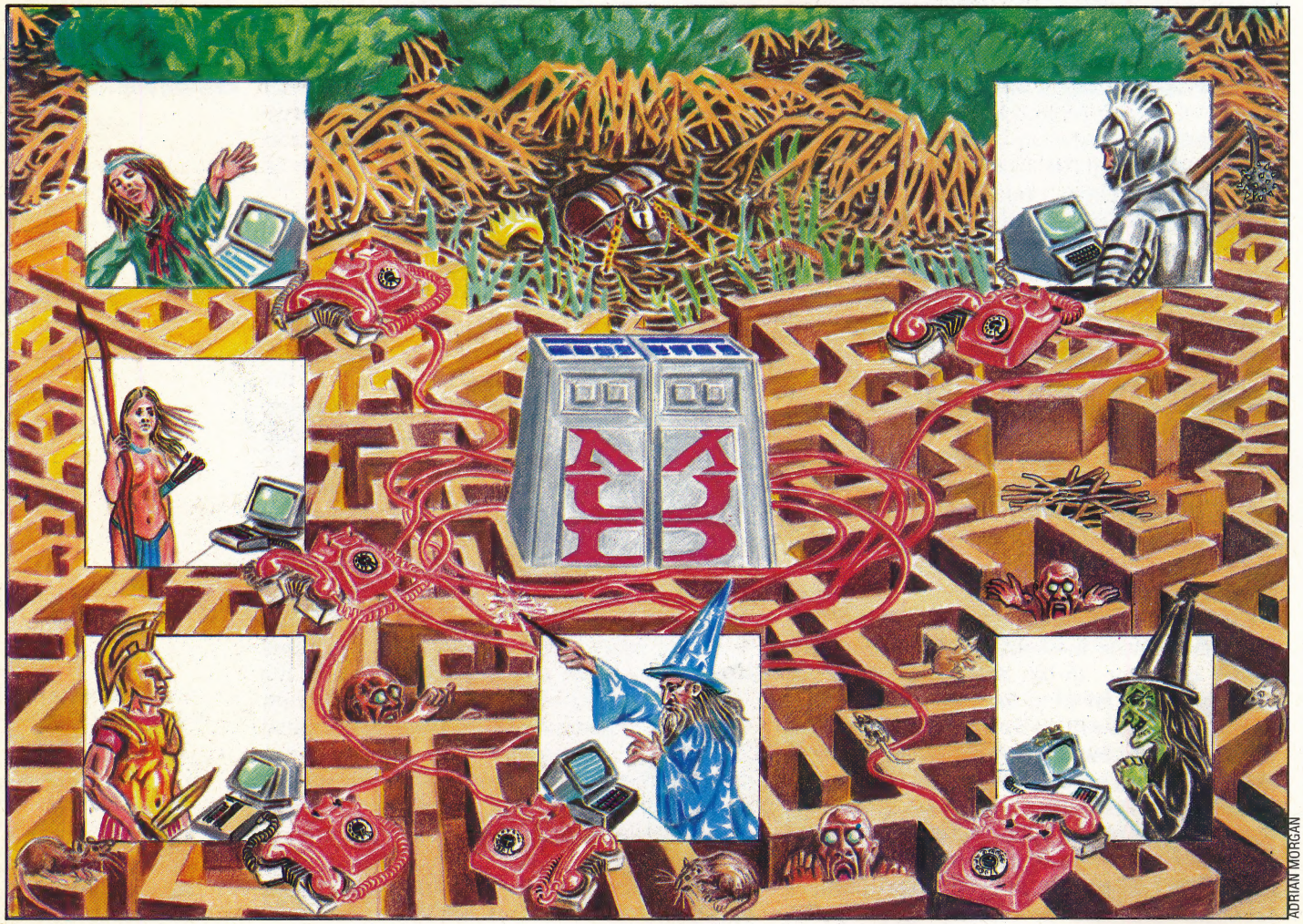
Wrestling With MUD
Mainframe adventures allow the involvement of many players — in Multi-User Dungeon, up to 43 Novices, Warriors, Enchantresses, and so on, compete or co-operate to gather treasure and become omnipotent Wizards or Witches. Mainframe adventures also support large detailed scenarios: MUD locations include The Land, various caverns, a forest, a dragon island, The Sea and The Swamp, all of which may contain treasure, goblins and zombies. Six telephone lines permit connection to home micro players

MUD is a real-time adventure where you meet other players, have conversations with them, ask their advice, join with them against a common enemy or fight them. They are not part of the program, they are playing the game at the same time as you, and so their actions affect yours.

MUD runs on a giant DEC10 computer at Essex University, and all you need to play it on your own microcomputer is a terminal emulator program, a telephone, a modem, and a Packet Switching System (PSS) account. A terminal emulator program allows your micro to communicate with the mainframe via a telephone link. You may choose to write your own emulator or to buy one. It should ideally allow the screen to scroll, and give an 80-character line length. Micronet software is not suitable since it does not permit this. It is possible to use micros with less than 80-column displays, but this is awkward. There are some excellent packages available; for example, Termi and Communicator are ROM chips for the BBC Micro.

The modem needs to be able to communicate with British Telecom's PSS, and can be 300/300, 1200/75, or 1200/1200 baud. A Micronet acoustic coupler is quite satisfactory. PSS is a networking service that enables you to contact distant host computers, often for the cost of a local phone call. There is also a charge for data sent over the PSS. For details contact British Telecom.

The procedure for gaining access to the mainframe is straightforward. With the terminal



ADRIAN MORGAN

software running, you dial the PSS exchange at which you are registered, connect your modem and key in your identity. Then you enter the PSS address of Essex University's DEC10. A special account has been set up at Essex to permit free access to MUD from midnight to 7am on weekdays, and 10pm to 7am at weekends, when the mainframe is under less pressure. Log in to this special account, and then enter RUN MUD.

The program is continually updated, so the first thing you will see is the date of the latest version. You choose a character name to play under, known as your 'Persona', and you tell MUD what this is. If you are playing the game for the first time then a new persona will be created for you, and you will be classified as a Novice. Levels of experience are:

Level	Points	Male	Female
1	0	Novice	Novice
2	400	Warrior	Warrior
3	800	Hero	Heroine
4	1600	Champion	Champion
5	3200	Superhero	Superheroine
6	6400	Enchanter	Enchantress
7	12800	Sorcerer	Sorceress
8	25600	Necromancer	Necromancess
9	51200	Legend	Legend
10	102400	Wizard	Witch

If you end your session without being killed the points that you have accrued will be noted by the program, and you can continue next time from that score. You get points and increase your level of experience by dropping items of treasure in the swamp, by overcoming the odd problem, by destroying various nasties like rats and zombies, and by winning a fight against another player. If you get killed in a fight with another player your persona is removed from the game, and you must begin again as a Novice, with no points. Therefore, it is always wise to consider alternatives to combat, especially as your opponent may have a stronger weapon or more stamina than you.

You start the game on a 'Narrow road between lands'. Typing 'WHO' will provide a list on the screen of all the others who are playing at that time. You may choose to say hello to one of them. For example, typing 'Jez, hi there – I'm new and could use some advice' would convey that message to the terminal of the player called Jez. You could talk to all the players at the same time by typing 'Shout, OK you terminal junkies, watch out cos here I come', but it is not advisable to start your first game in this way.

Typing 'HELP' will give you some information about how to move and will give a brief explanation of many of the commands. Movement choices are explained in this way: 'Most simple movement commands are allowed, e.g. n, sw, west, up, jump, plus others you'll have to find out!'

The commands available may be listed by typing 'COMMANDS'. There is quite a lot of information available, and scribbling a copy on paper while the words scroll down the screen is not a realistic option. Some terminal software will allow you to copy everything that appears on your screen onto disk for you to look at later. This will also enable you to construct a proper map of the land, which can be amended after each game. These are the commands that were available on one day:

COMMANDS (abbreviation in upper case)

AutoWho,‹seconds›	Back	BERSERK
BRIEF	BUG	BYE
CONVERSE	DRop ‹item›	DRop ALL
EMPTY ‹bag›	EXITS	Flee‹direction›
FOLLOW ‹name›	Get ‹item›	Get ALL
GIve ‹item› TO ‹name›		go ‹direction›
Help	HELP ‹name›	HINTS
HouRS	HUG ‹name›	INFO
Inventory	KEEP ‹item›	Kill ‹name›
KISS ‹name›	LEVEL	LOG
‹level›, ‹message›	Look around	Look ‹bag›
Look ‹direction›	LOSE ‹name›	MEDITATE
NoPassWord	PassWord	PERSONA
ProNouns	QuickWho	Quit
"‹message› "	REFUSE ‹name›	
RETaliate ‹item›	SAVE	SCore
SHout,‹message›	SLEEP	SPELL
STeal ‹item› From ‹name›		
tell ‹name›,‹message›		UNKEEP
VERBOSE	WEIGH ‹item›	WHEN
WHO	WRITE ‹object›, ‹message›	

MUD is a large text-based adventure, with lengthy and detailed descriptions of the locations. When you are familiar with the scenario you can type 'BRIEF' and not have to read the descriptions each time. Prestel subscribers will be aware of the slowness and inadequacy of teletext graphics, and whilst graphic-based adventures are an interesting novelty, dedicated adventure games players will always choose a text-based game. A text-only adventure allows a greater imaginative involvement than a graphic one, in the same way that a radio play can be more enjoyable than one on the television. Another disadvantage of a graphic adventure is that each make of home micro will need a different version of the game, because of the different graphic capabilities of home computers. MUD players are likely to be using a BBC micro, an Apple or a Spectrum, but others will have second-hand terminals from junk shops, and therefore the range of machines using the program is extensive.

MUD is expected to be marketed soon. The program's original authors, Richard Bartle and Roy Trubshaw, are writing a version to run on a VAX Computer, which will be marketed by Century Communications. (A VAX plus software and disk drives costs around £50,000.) The game will still be played via telephone links, like PSS and Prestel, although with enough demand it could well be available on cable, as well. MUD players will then pay a fee to join the game, perhaps £10 or £15 per quarter, plus a small charge for each hour they play.

Dungeon Master

For further details about the Multi-User Dungeon contact:

Richard Bartle
Department of Computer Science
University of Essex
Colchester
Essex

PRIMITIVE PARTS

An algorithm is a series of instructions that describe how some process may be performed. A knitting pattern is an algorithm; so is a recipe; and so is a computer program. We discuss how an understanding of the principles of algorithms can improve your programming.

An algorithm describes a process either in terms of other processes that have already been defined, or in terms of processes that are so basic that they do not need to be defined. Thus, in a recipe, one instruction may be 'prepare a bechamel sauce', where a bechamel sauce recipe (algorithm) has been given elsewhere in the cookbook. Another instruction may be 'bring the mixture to the boil', where the operation of bringing something to the boil is assumed to be fully understood by the user. In programming terms, algorithms are constructed from instructions that either use algorithms (procedures, routines, functions) written elsewhere in the program, or ones built into the language (commands such as PRINT and DIM, or maths functions like LOG and TAN).

This article looks at how algorithms are constructed from other algorithms and *primitive* processes. The primitives at the disposal of a programmer are the commands and functions in the language. From these, algorithms are written that can do small things (move a sprite, say, or accept a number as an input). These algorithms are then used to build more general algorithms (updating the game display, or controlling a menu system), and these algorithms are in turn used as parts of larger ones again until the whole program, viewed as a single algorithm, is written in terms of lower-level algorithms. This concept is the basis of what is known as *structured* or *modular* programming and is a subject we will return to later in the course.

DESIGNING ALGORITHMS

An algorithm has an input and an output. This is just to say that, as a process, the algorithm will work on some initial data to produce a result. This initial data is passed to the algorithm from outside in the guise of 'parameters', which remain constant for any particular use of the algorithm but may change between different uses. Passing parameters will be familiar to even a novice programmer since the simple program:

```
10 PRINT "Hello World!"
```

passes the parameter 'Hello World!' to the algorithm called by the PRINT command. Similar examples are FNA(P), TAN(P), LEFT$(P$,5) and POKE P,5, where P, P$ and 5 are all parameters. In the same way, an algorithm's results are passed back as parameters. If the programming language being used has local variables (e.g. PASCAL and C) then the parameters would normally be passed with a procedure call, as in:

```
procedure(parameter1,parameter2,etc.);
```

It is an essential first step in designing an algorithm to consider the contents of the input and output parameters, their types (integer, floating point, real, string, etc.) and their magnitudes and ranges.

When the input and output of the algorithm have been defined, the next step is to form ideas as to how one can be transformed into the other. Unfortunately, there is no 'cookbook' method for creating these transformations as they require creativity and ingenuity. However, there are several ways of helping the process along.

The most obvious and most often overlooked is to borrow the algorithm from somewhere else. At the simplest level, the in-built functions of a programming language provide many useful algorithms, such as string-handling, trigonometric functions, input/output, and (possibly) sorting and matrix manipulation. Apart from this, the algorithm needed may already exist in another of your programs. The code for this could be incorporated in the new program (creating your own library of algorithms is extremely useful). In addition, there are published collections of algorithms that are often available from public libraries. *The Art of Computer Programming, Vol I: Fundamental Algorithms* by D. E. Knuth, although heavy going, is highly recommended.

Programs published in the computer magazines are well worth scrutinising for routines that may be of use. Finally, there are algorithms that are used in other pertinent domains, and although they were never meant for computing they are extremely useful. The accountancy section of the local library will be full of books containing formulae for calculating balances and depreciation. A little research among these could make writing an accounts program a lot easier and the result is likely to be a lot more reliable. The same is true for other disciplines: engineering, electronics, maths, etc.

Whether adapting an existing algorithm, or creating one from scratch, there are certain criteria that must be applied to each instruction it contains. These are *definiteness* and *effectiveness*. Definiteness means that the instruction should

not be ambiguous in any way. Ambiguity is easy to introduce at an early stage when the algorithm is being written down in English. Words like 'and' and 'or' in English are very different from the AND and OR of Boolean logic. For example, if the algorithm was meant to select all the names in a list that begin with an 'A' *and* all those beginning with 'B', you could easily write code like:

```
IF FIRSTLETTER="A" AND FIRSTLETTER="B" THEN
```

which is wrong because a logical OR is needed!

The criterion of effectiveness is a demand that the program should not contain impossible instructions. An instruction is said to be effective if it can be done with a pencil and paper in a finite time. This means that instructions like 'let X equal the highest prime number' are not effective (there isn't a highest prime number).

GENERAL CONSIDERATIONS

There are also criteria to judge the algorithm as a whole. An algorithm must *terminate*. The algorithm that follows does not terminate (even though its instructions are definite and effective) and if this was coded into a program it would endlessly loop:

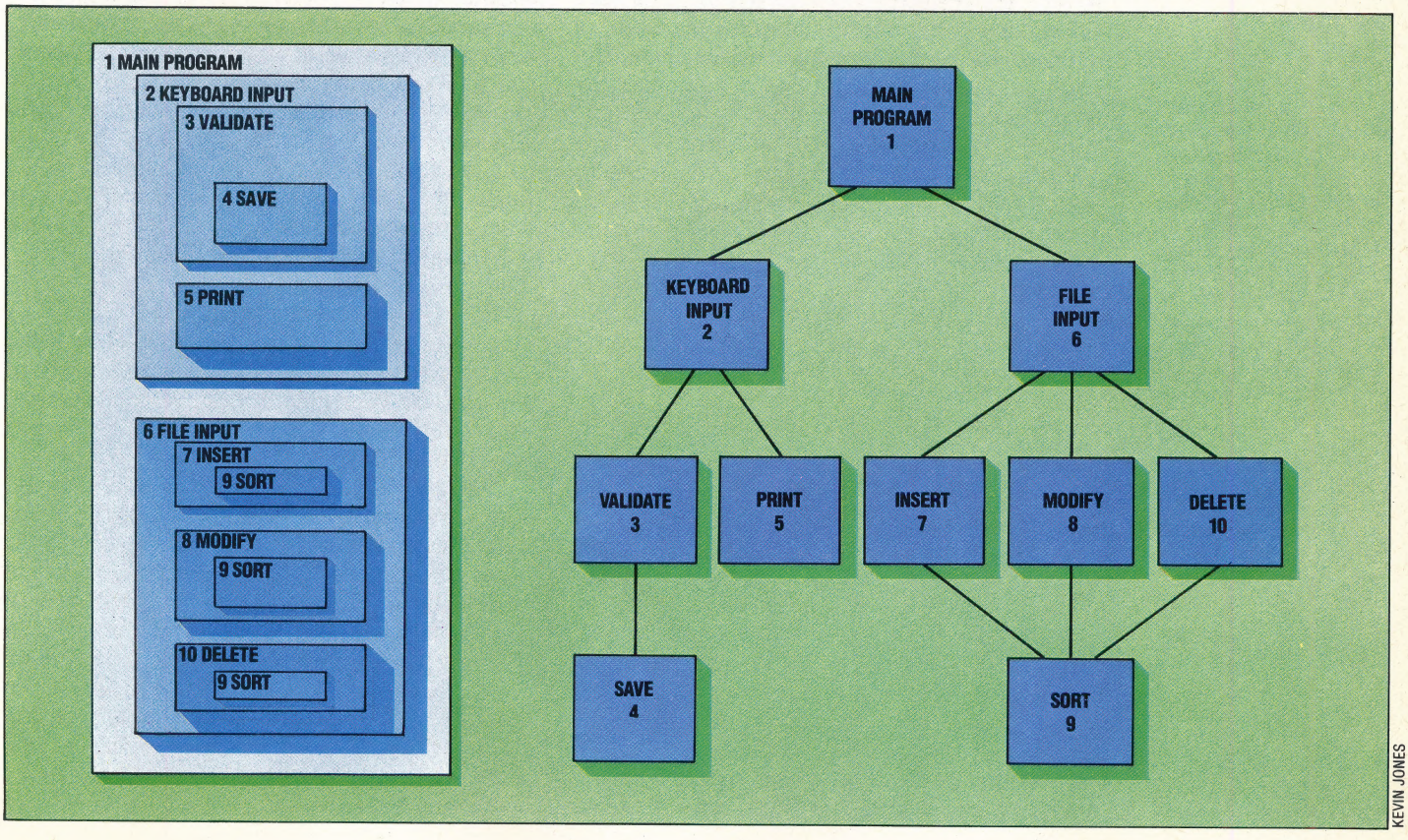
```
step 1 let I equal 1
step 2 if I > 3 then exit
step 3 goto step 1
```

Telling whether an algorithm will terminate is not always easy, but, in general, algorithms that involve loops test for a particular condition before they terminate (e.g. if I > 3 in our example), and it is necessary to check that it is possible to meet that condition.

Efficiency, generality and *elegance* are ways of judging between different algorithms. Efficiency is usually judged in terms of time and memory use. The two are usually quite compatible — fast code may need relatively little space, but bear in mind that this need not be so. Having found an algorithm, it can be 'tuned' for efficiency by changing its details. A calculation will be noticeably faster and will use less memory if, for example, integer rather than floating point arithmetic is used. Alternatively, a completely different algorithm for doing the same thing could be found.

Generality is the ability of an algorithm to cope with many different situations apart from the one for which it was designed. It is worth while, in the long run, to attempt to make all algorithms as general as possible. If a program called for a yes/no response several times, it would be worth writing a routine that prompts the user with 'please type y or n', accepts the input, checks whether it is 'y' or 'n', reprompts if it is neither and otherwise returns the appropriate response. However, the routine could be made more general if it could be fed with different prompts and potential replies, so it could be used in many different situations. Elegance means finding algorithms that are both simple and ingenious. In all cases it is more sensible to find efficient, general algorithms rather than elegant ones.

Another important aspect of algorithms is the flow of control and of data within them and how this can be represented with flow charts. This is the subject of the next instalment in this series.



KEVIN JONES

Pyramids And Primitives
The block-structure diagram on the left clearly shows the nesting of a program's algorithms, while the procedure flow diagram on the right emphasises the articulations and process levels of the same program. The most 'primitive' algorithms are the most deeply nested, and the lowest in the hierarchy

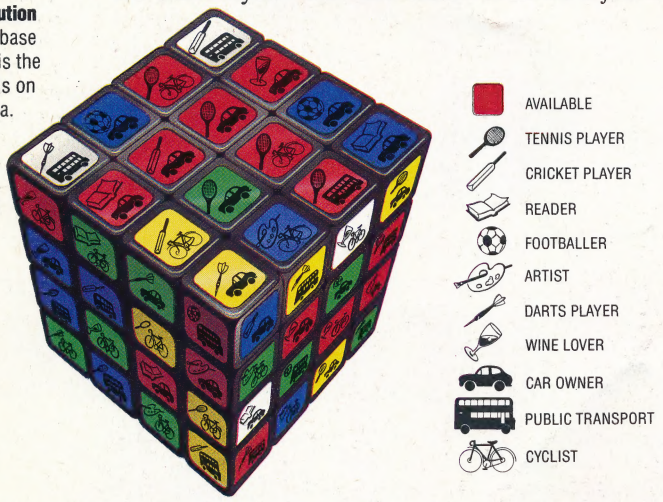
DAISY WHEEL

Until recently, microcomputer owners requiring letter-quality printing were restricted to one variety of printer — the *daisy wheel*. The name derives from the printing mechanism: individual letters are mounted on plastic spokes or 'petals' arranged in the shape of a wheel about three inches (7.5 cm) in diameter. The wheel spins until the desired character is at the top, and then a hammer strikes the 'petal' against the ribbon, making an impression of the character on the paper. Electronic typewriters also use this technique. In some models the letters are mounted in a 'thimble' shape.

Daisy wheel printers are considerably slower than their dot matrix counterparts; a £400 model can print only 10 to 15 characters per second (cps), as opposed to typical dot matrix speeds of 100-160 cps. The fastest daisy wheel models can print at 60 cps, but these are priced at around £2,000. The main restriction on speed is the fact that the printer motor must constantly accelerate and decelerate while turning the wheel. Close examination of a daisy wheel may give the impression that letters are arranged at random, but the layout is designed to reduce the average amount that the wheel must turn when printing standard English text.

DATABASE

A *database* is a general-purpose software package that is designed for the easy storage and retrieval of data. All the information about one person (in a mailing list) or object (in cataloguing or stock control applications) is contained in a *record*, which corresponds to a card in a card index file. Each record consists of *fields* of information, each with its own field name. For example, the field named 'Price' in a stock control program may contain the data '£34.25' in one particular record. A collection of records sharing the same layout and field names is called a *file*.



Searching For The Solution
The essence of a database management system is the ability to select records on multiple search criteria. The Rubik Cube represents the database, and its facets are the records

Simple databases restrict the user to work on a single file at a time, but multi-file packages allow data to be transferred between different types of file. In an invoicing application, for instance, information from a customer file that contains the customer's address and credit limit may be merged with data from a product file that gives details of prices, product codes and descriptions. The most sophisticated microcomputer databases are programmable: the package can 'learn' sequences of commands that are used frequently and will execute them at a single keystroke. Such packages are also referred to as *application generators*.

DATA COMPRESSION

The rule of thumb method for evaluating data storage requirements is to allow for one eight-bit byte per character, as well as making some allowance for operating overheads such as file header information and control characters (see page 348). *Data compression* involves trying to improve on this rate for two purposes: increasing the effective capacity of a disk drive, and reducing the time needed to transmit data over a distance.

At first this may seem very difficult to achieve, especially as all characters and some commands are ASCII coded. Consider, however, messages that use just upper case text and a maximum of six other punctuation symbols — just five bits would then be needed to represent the 32 (2^5) possibilities. Characters will no longer correspond one-to-one with the byte locations, and a special program must be employed to compress the data before recording on disk, and then decompress it again for use by an application program. Such utilities (also called *compactors*) can be purchased to work on standard CP/M files. An additional benefit is some measure of security — compressed files can't easily be read on another system.

Compression rates greater than eight-to-five rely on frequency analysis — some sequences of characters occur more frequently than others. In normal text, a standard list of the 100 most common words (and, the, etc.) will account for half the text. If these words are replaced by a single byte, with a flag-bit to differentiate them from normal characters, the saving will be considerable. This is called *tokenising*.

DATA CORRUPTION

Data corruption is most likely to happen when writing onto a magnetic surface (because of imperfections in the media), or when transmitting data over a distance (because of extraneous noise). It can happen to data residing in memory, if an individual RAM chip should overheat and behave unreliably, or there is strong electrical interference from nearby.

Magnetic media should always be handled carefully, but the only way to guard against corruption generally is to employ the principle of *redundancy* — the opposite of data compression. This means taking more space or time to represent a single piece of data than is strictly necessary. Parity and Hamming codes are the commonest means of safeguarding against data corruption.

THE SMART SET

Coleco is an American company that made its name as a manufacturer of television games consoles. Recently, however, it moved into the home computer market with the Adam, a machine that boasts 80 Kbytes of RAM, a built-in word processor, a tape data storage system, and a daisy wheel printer. All for £625.

The Adam system is based around the ColecoVision television games console, and owners of this unit may upgrade it to full Adam specification with add-on units. In fact, all Adams on sale at the present time are based on the ColecoVision module, although a more compact, stand-alone Adam is planned.

When first unpacking the boxes containing the Adam system, the sheer number of components is a little unsettling. The ColecoVision module clips into a large plastic tray, and the 'memory module' (which also contains the Z80A microprocessor and the tape drive) connects to the games unit via an expansion port. The module is held in place by retaining clips on the tray. The keyboard is connected by a length of coiled cable, and one of the two games controllers supplied may be slotted onto the side of the keyboard to act as a numeric key pad. The printer is connected to the memory module, and the power is supplied to the whole system via a mains lead to the printer.

The keyboard has 75 keys, including six function keys, a set of word processor control keys, a cursor cluster, and the usual alphanumerics. It is well designed and light, so it is easy to use in a lap-held position.

When the system is first switched on, the Adam is in 'electronic typewriter' mode. Pressing a key displays the character on the screen and at the same time it is printed out. The screen is a representation of a sheet of paper and typewriter platen. The usefulness of the Adam in this mode is limited, but the press of a key switches the system to 'SmartWriter' mode.

SmartWriter is a simple but effective word processor that is obviously designed with the beginner in mind. The 'typewriter platen' display is retained, and the bottom line of the screen displays the commands allocated to the function keys. Sensible use of these keys, together with the various word processor control keys on the keyboard, makes SmartWriter operation extremely easy — in fact the manual is largely superfluous as the menu options lead the user through all the steps necessary to store, display and print text. The screen format is 36 columns by 20 lines, but longer lines are dealt with by using the screen as a 'window' on the whole text.

The tape drive is similar in principle to Sinclair's Microdrive, but has a larger capacity (a claimed

CHRIS STEVENS

The Adams Family
The Adam is an all or nothing machine. It is sold as a complete system: micro, high-speed tape drive, daisy wheel printer, keyboard, two joysticks and three pieces of software (two excellent games and a word processor.) This makes the Adam quite a powerful home computer, yet it is essentially an upgrade of a TV games unit

Pass The Buck
The Buck Rogers game comes on tape for the Adam's own tape drive and makes full use of the drive. The game is split into several stages that are too large to fit into memory at once, so instead the Adam loads one stage from tape and then while that is playing it loads the next, and so on. At the end of each game the tape automatically re-winds, and if a high score has been achieved it is recorded onto the tape. Thus, the high score table shows the best scores ever managed, not just the best since the computer was turned on

256 Kbytes per tape). It is much faster than ordinary cassette tape. It is also under full computer control, so the tape can be rewound by the computer to find a particular file. Tapes (or 'data packs' in Coleco-speak) are supplied ready-formatted, so ordinary audio cassettes may not be used. New tapes cost up to £5. When the Adam is switched on, the program on any tape present in the drive is loaded — this may be a game pack, the SmartBASIC cassette or a program of the user's own. If the drive is empty, the system defaults to 'electronic typewriter' mode.

The daisy wheel printer uses either fanfold or single-sheet paper, and gives a print quality that is comparable to an electric typewriter, but it is extremely slow and noisy.

Despite the 80 Kbytes of RAM, the cassette-based SmartBASIC leaves only 28 Kbytes for the user. SmartBASIC itself is similar to Apple's version of BASIC, and so is easy to use, with good graphics and a choice of 16 colours. The BASIC manual, however, is truly abysmal, written in a condescending tone and full of twee expressions, such as 'boo-boo' for 'error'. But the similarities between SmartBASIC and Apple BASIC means that users have a large number of books and other material to refer to.

At first sight the Adam system appears to offer excellent value for money. The built-in software is ideal for writing letters, reports or short articles, and the tape system is simplicity itself. An added bonus is the large amount of available high-quality games software — not only may ColecoVision cartridges be used on the Adam, but the purchase of an adaptor will make a huge range of games from Activision and Atari available. More serious software is being sold on tape, although the number of packages is rather limited. They

Detached Keyboard
Only two home micros are sold with detached keyboards even though these are considered essential for business computers. The Adam keyboard includes function keys and keys dedicated to the word processing software that comes with the micro

Joystick Handle
Although the arm of the joystick handle is much shorter than most joysticks, it still gives re

Joystick Storage
Two joysticks can be stored here when they are not being used. The joysticks plug into sockets at the side of this area

ColecoVision Video Games System
This unit is sold by itself as a dedicated games computer, but can be expanded into a home computer by adding the other units

Equipment Tray
This fits onto the bottom of the two units to hold them firmly together

Printer Connector
The printer included with the Adam plugs into this socket. Power is supplied to the Adam through this, so the printer must always be connected, even if it is not being used

Digital Tape Drive
This may look like an ordinary cassette player, but it is a high-speed tape unit under full computer control

CHRIS STEVENS

CHRIS STEVENS

Joystick Fire Buttons

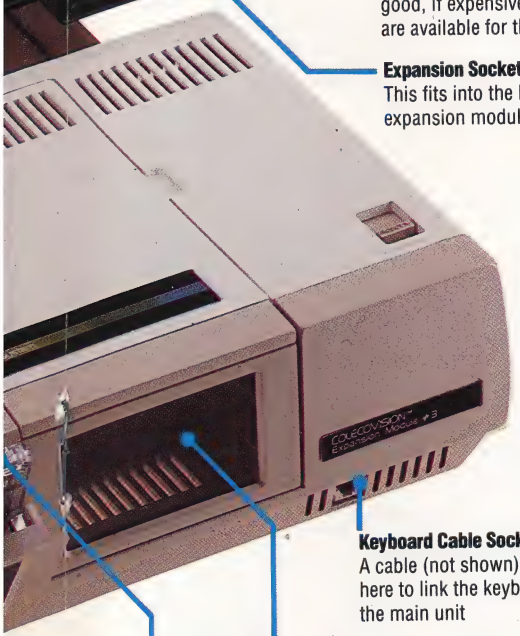
Joystick Number Pad
The pair of joysticks that come with the Adam are unusual in having number pads built-in

Joystick Holder
This clips onto the keyboard and provides a space to rest a joystick

Reset Button

Games Cartridge
This plugs into a slot in the games unit. Several dozen good, if expensive, cartridges are available for the machine

Expansion Socket
This fits into the back of the expansion module



Keyboard Cable Socket
A cable (not shown) plugs in here to link the keyboard to the main unit

Space For Second Tape Drive
These are not yet available

Digital Data Pack
Not to be confused with a blank tape cassette. The Adam has to use special pre-formatted tapes which cost nearly £5 each, and store 256 Kbytes per tape

Adam's Joysticks
Two joysticks are supplied as standard with the Adam. One of these doubles as a numeric keypad and slots into a special holder alongside the keyboard. Coleco also markets two 'deluxe' games control systems, at £50 each. The Super Action Controller set comprises a pair of easy-action joysticks with numeric pads and rifle-style triggers replacing the normal Fire buttons. These are fitted with standard Atari-type D-connectors, and so may be used with other systems. A free Baseball games cartridge is included.

The Roller Controller is an arcade-style games control panel. This has two standard Coleco joysticks and a 'trackball' controller for fast response and accurate control. Two sets of dual fire buttons allow two-player action, and a mode switch selects either joystick or trackball operation. Supplied with this unit is Slither, a Centipede-style arcade game.

include a spreadsheet, database, revision aid, and a version of the LOGO language. The addition of the promised 144 Kbytes memory expansion should give CP/M capability of a kind, and even disks are planned.

But there are drawbacks. The print quality is not as good as one might expect from a daisy wheel, the printer is woefully slow and noisy, there is a severe lack of user RAM, and the fact that BASIC must be loaded from tape soon proves annoying. With the recent drop in daisy wheel printer prices, it should be possible to put together a system that out-performs the Adam but costs approximately the same. That said, the Adam would appear to be well worth buying if the user already possesses the ColecoVision console. For those who wish to purchase a home computer system with word processing capabilities, the Adam is certainly worth considering — but there may well be other, better alternatives.

COLECO ADAM

PRICE

£525 (expansion module)
£625 (including ColecoVision games console)

DIMENSIONS

381×279×102mm (expansion memory unit)
381×355×152mm (printer)
381×152×51 (keyboard)

CPU

Z80A

MEMORY

80 Kbytes of RAM, of which 16K is used for video display. Expandable to 144K with optional RAM pack

SCREEN

36 columns × 20 rows (31 columns × 24 rows in BASIC)
256 ×159 pixels in high resolution with 16 colours

INTERFACES

SmartWriter printer connector, Adam Net modular phone plug, three expansion slots, ColecoVision expansion interface

LANGUAGES AVAILABLE

SmartBASIC supplied on cassette; CP/M operating system

KEYBOARD

75-key, QWERTY layout, sculptured typewriter-style keys. Includes 6 programmable function keys

DOCUMENTATION

Three manuals come with Adam: 64-page Set-Up manual, a guide to the SmartWriter word processing program, and a SmartBASIC programming manual. The documentation is generally easy to follow, but a bit wordy and insubstantial. The BASIC manual is incomprehensible

STRENGTHS

Wide availability and high quality of ColecoVision games cartridges make Adam ideal for games players. Word processing software is easy to use

WEAKNESSES

Slow, noisy printer. Data packs available only from Coleco. Limited software available on data packs

TREAD LIGHTLY

In this short series of articles we shall be looking at the construction of a graphics game using BBC BASIC. The game is designed to run on the Model A, Model B and the Electron. As each phase of the game is developed, the appropriate section of the program will be listed, allowing you to build up the game with each instalment.

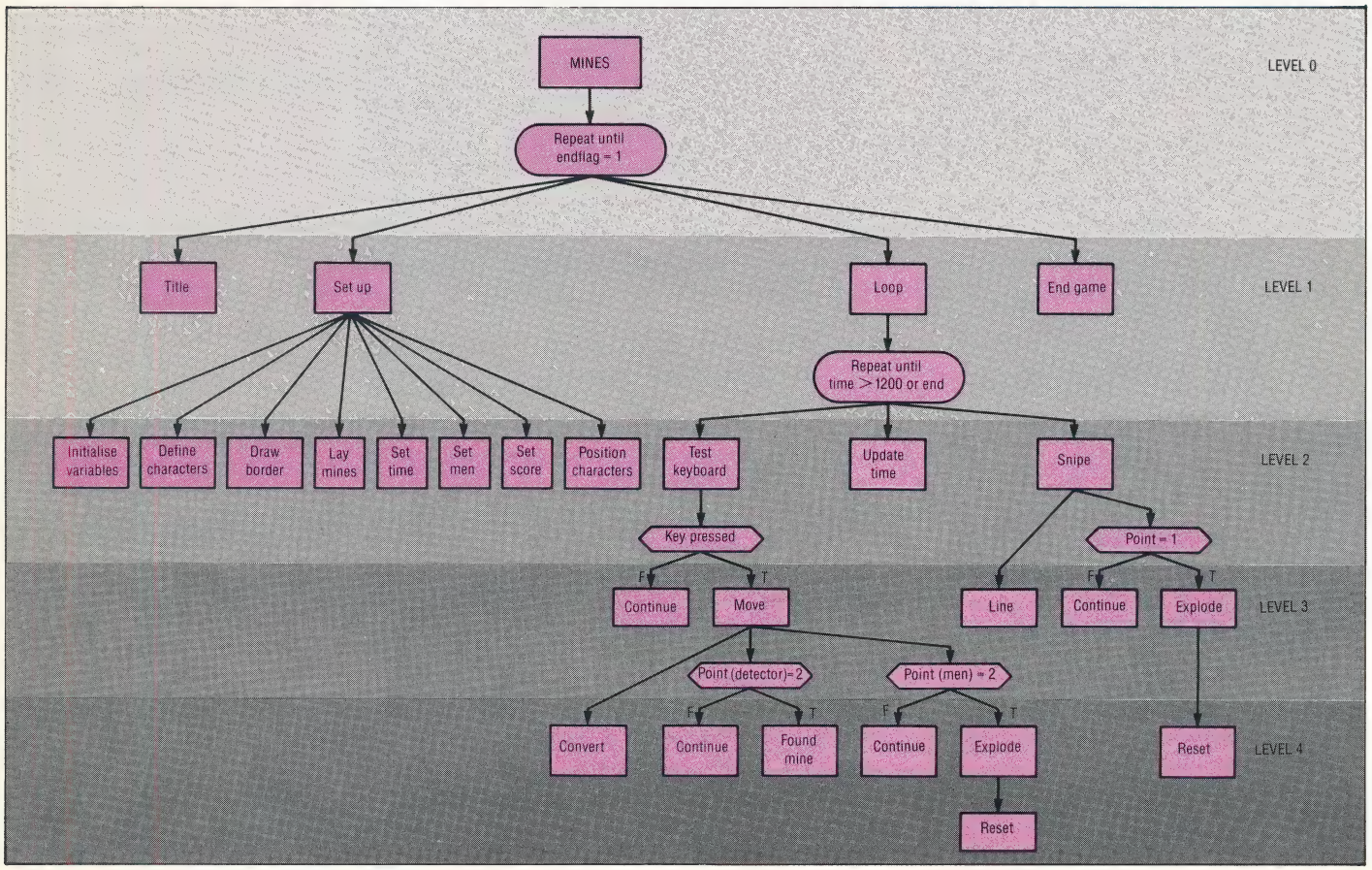
Routine Procedure
Unlike a flow chart, this structure diagram shows the procedural structure of the program rather than the flow of control through it. A capsule indicates the start of a REPEAT... UNTIL loop; the lozenges are decision boxes — when the test fails, the enclosing loop continues. The Level numbers show the program's block structure: all loop starts and procedure calls open a new block of program and a lower logical level — compare this with the diagram on page 387

BBC BASIC has two major advantages for the programmer over 'standard' Microsoft BASIC: it is fast in execution and has features that allow you to structure programs. The essence of developing a structured program is to develop small, independent sections of code that can be individually debugged before assembly into a larger program. Any BASIC program can be structured to a certain extent by the use of subroutines to code each module of the program, but BBC BASIC has special types of subroutines, known as *procedures*. These can be thought of as blocks of code that are designed to do a specific job within a program. For example, let us imagine a piece of program that has to pause between each instruction for a given time. In standard BASIC this may be written using a *dummy loop*; that is, a loop that does nothing except take time to execute:

```
10 PRINT "FIRST SECTION"
20 FOR I=1TO100: NEXT I
30 PRINT "SECOND SECTION"
40 FOR I=1TO100: NEXT I
50 PRINT "THIRD SECTION"
60 FOR I=1TO100: NEXT I
70 PRINT "FOURTH SECTION"
80 END
```

A better approach, however, would be to place the delay loop in a subroutine:

```
10 PRINT "FIRST SECTION"
20 GOSUB100
30 PRINT "SECOND SECTION"
40 GOSUB100
50 PRINT "THIRD SECTION"
60 GOSUB100
70 PRINT "FOURTH SECTION"
80 END
100 REM ** SUBROUTINE **
110 FOR I=1TO100:NEXT I
120 RETURN
```



Replacing the subroutine with a procedure in BBC BASIC gives the following code:

```
10 PRINT "FIRST SECTION"
20 PROCdelay
30 PRINT "SECOND SECTION"
40 PROCdelay
50 PRINT "THIRD SECTION"
60 PROCdelay
70 PRINT "FOURTH SECTION"
80 END
100 REM ** DEFINE PROCEDURE **
110 DEF PROCdelay
120 FOR I=1TO100: NEXT I
130 ENDPROC
```

There are a number of similarities between the subroutine construction and the procedure construction; for instance, both come after the END statment but can be called repeatedly from within the main program. The principal advantage of the procedure is that it is called by name rather than by line number. The DEFinition of the PROCedure could start anywhere after the END statement.

If we wanted our example program to be able to wait for different lengths of time before each section, then a more important advantage of procedures can be utilised: the ability to pass values into a procedure definition. Let us say that we wished the pause between the first and second sections to be 100 loops, the pause between the second and third sections to be 200 loops and the pause between the third and fourth sections to be 175 loops. In standard BASIC the value of I would need to be assigned each time before calling the subroutine. Using procedures, the value can be passed by means of a bracket at the end of the calling statement:

```
10 PRINT "FIRST SECTION"
20 PROCdelay(100)
30 PRINT "SECOND SECTION"
40 PROCdelay(200)
50 PRINT "THIRD SECTION"
60 PROCdelay(175)
70 PRINT "FOURTH SECTION"
80 END
100 REM ** DEFINE PROCEDURE **
110 DEF PROCdelay(N)
120 FOR I=1TON: NEXT I
130 ENDPROC
```

More than one value can be passed for use within a procedure if the items are separated by commas. Variable names can also be used in this way to pass the value of the variable at the time when the procedure is called.

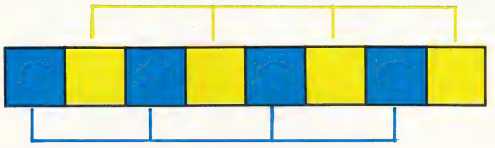
The game we shall be constructing is for one player and uses the keyboard cursor control keys to move a mine detector around a minefield. Points are scored for each mine successfully dealt with. There are, however, several things to slow down your progress around the minefield. The chief consideration is your assistant who mirrors your every move. As you go round dealing with the mines you must make sure that he doesn't tread on one. You are also being sniped at and there is a two minute time limit to the game. In the final version, you will have four willing assistants per game and a choice of skill factors from 0, the easiest, to 9, the most difficult.

Because BBC BASIC uses procedures, flow charts aren't of much use. Instead, a *structure diagram* can be used to illustrate what procedures are required and how they fit together to make the final program. REPEAT . . . UNTIL loops are shown as 'sausage' shapes in our diagram. Decision boxes are like the more usual diamond-shaped box but with the top and bottom cut off to save space. It should be stressed that this diagram was not drawn out in its entirety before programming started, but evolved out of a series of refinements.

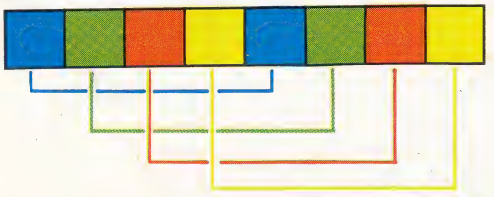
SET-UP PROCEDURES

Brfore we can start defining routines to place objects on the screen, we must decide on the screen display mode we are going to use. There are three main factors to take into account: resolution, colour and memory. In general terms, the more information the screen has to hold, the more memory it will require. So, higher resolutions and more colours mean more memory. If your program is short this may not be important to you, but the program we are designing is fairly lengthy. We do require a few different colours to distinguish the mines and the detector/assistant and to make the game visually appealing. On the Model B we are offered two medium-resolution modes. Mode 2 gives us 16 colours to play with, whereas mode 5 offers only four. By looking at how the BBC interprets bit patterns in each mode we can see why mode 5 uses substantially less memory than mode 2.

Unlike some micros, the BBC holds colour and pixel on/off information within one byte for each small area of the screen. In mode 2, four bits are required to represent the 16 different possible colours. Thus, one byte can hold information about the colour of two pixels only. The bits within the byte are arranged as follows:

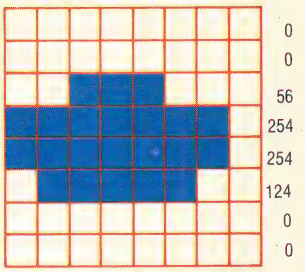


As mode 5 restricts itself to four colours, only two bits are needed to hold the colour information. Thus, one byte can represent four pixels as shown:

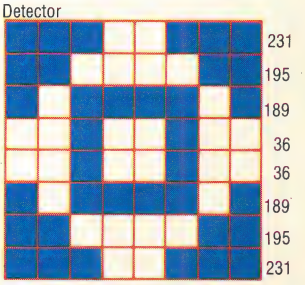


Both modes have 160 by 256 pixel resolution, hence the number of bytes required for mode 2 screen memory is (160×256)/2=20 Kbytes,

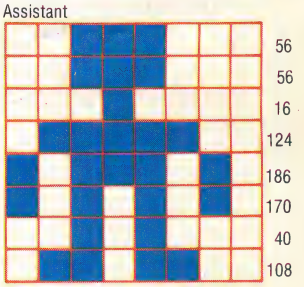
Mine



Detector



Assistant



Picture Points
The user-defined characters representing the mine, the detector and the assistant are described. The pixel information is divided into eight bytes, with each byte being the binary 'picture' of one row of the shape

whereas mode 5 needs (160×256)/4=10 Kbytes of memory. By choosing mode 5 we allow ourselves an extra 10 Kbytes of memory for our program. An added bonus is that mode 2 is not available on model A machines, but mode 5 is!

The construction of user-defined characters is simple on the BBC. Each character is made up of eight numbers representing the decimal equivalents of each row. The character designs for the mines, detector, and assistant are given.

The command VDU23 allows the programmer to define characters with ASCII codes from 226 to 255. The second number in the VDU command denotes which code you wish to assign to the shape defined by the last eight numbers.

```
VDU23,224,0,0,56,254,254,124,0,0
```

defines CHR$(224) as the mine shape. To print this character we simply issue the instruction PRINT CHR$(224). The following procedure defines the three characters to be used in the game.

```
2380 DEF PROCdefine_characters
2390 REM ** MINES **
2400 VDU23,224,0,0,56,254,254,124,0,0
2410 REM ** MINE DETECTOR **
2420 VDU23,225,231,195,189,36,36,189,195,231
2430 REM ** ASSISTANT **
2440 VDU23,226,56,56,16,124,186,170,40,108
2450 ENDPROC
```

SCREEN LAY-OUT

Once the shapes have been defined we can print them onto the screen. This is most easily done using the PRINT TAB(X,Y) command. In mode 5 there are 32 rows, each with 20 characters. This means that X ranges from 0 to 19 and Y ranges from 0 to 31. The minefield is to occupy the area shown in the diagram.

To lay the mines randomly within the given area we can make use of the RND(N) command. If N is a whole number, then RND(N) returns a whole number between 1 and N. The horizontal co-ordinate of each mine must be chosen to lie between 2 and 17. RND(16) gives numbers from 1 to 16 so RND(16)+1 will select co-ordinates within the area of the field. This procedure will lay as many mines as specified by the value passed to it:

```
2560 DEF PROClay_mines(number_mines)
2570 REM ** CHANGE COLOUR 2 TO GREEN **
2580 VDU19,2,2,0,0,0
2590 FOR I=1 TO number_mines
2600 PRINTTAB(RND(16)+1,RND(25));CHR$(224)
2610 NEXT I
2620 ENDPROC
```

In mode 5 we are restricted to four colours and these are normally black, red, yellow and white, corresponding to colour numbers 0, 1, 2 and 3. However, we don't have to use these colours and can change them using the VDU19 command. The numbers 0 to 3 are known as the *logical* foreground colours. Each of the 16 colours on the BBC has a number that has nothing to do with the mode used. These are called the *actual* colour numbers, a table of which appears on page 224 of the user guide. Any of the four logical colours can be assigned one of the sixteen actual colours. For our game we wish the mines to be green (actual colour number 2). We do not want the yellow that is normally given to us as logical colour 2. The VDU19 command does this.

The detector and assistant take up their initial positions in the bottom-left and top-right corners, respectively. As we will probably want to re-position the detector and assistant later, the procedure for positioning them will use variables (xdet,ydet) for the detector's co-ordinates and (xman,yman) for the assistant's co-ordinates.

```
2830 DEF PROCposition_chars
2840 COLOUR 1
2850 PRINTTAB(xdet,ydet);CHR$(225)
2860 PRINTTAB(xman,yman);CHR$(226)
2870 COLOUR 2
2880 ENDPROC
```

The COLOUR commands at the beginning and end of the procedure select the logical colour that future text will have. COLOUR 1 selects logical colour 1 (red) to print the detector and assistant and COLOUR 2 restores the colour green for future printing. Before this procedure can be used, however, the values of xdet, ydet, xman and yman have to be assigned. This is done in another procedure, along with initialising some other variables that will be used elsewhere.

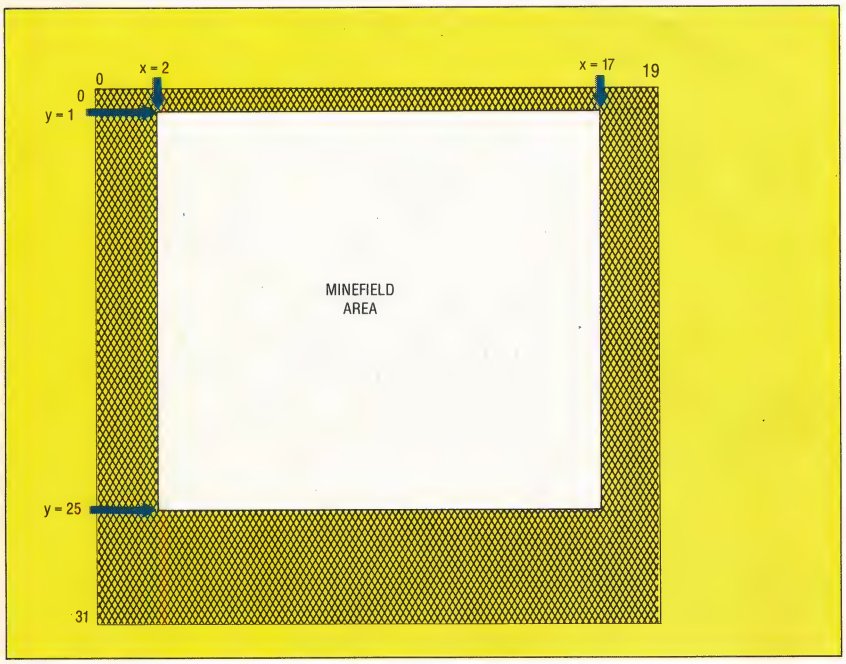
```
2320 DEF PROCinitialise_variables
2330 xdet=2:ydet=25:xman=17:yman=1
2340  xstart=120:xfinish=1144
2350 zero$="000000"
2360 ENDPROC
```

All these procedures can now be tested by a short calling program as follows:

```
 5 MODE 5
10 COLOUR 2
20 PROCinitialise-variables
30 PROCdefine-characters
40 PROClay-mines(40)
50 PROCposition-chars
60 END
```

In the next instalment we shall look at timing and controlling movement from the keyboard.

Danger Area
The minefield occupies 20 rows of 16 characters, and the mines are randomly 'sown' during the set-up procedure



JANET BARRANCE

ROCKET MAN

Jet Pac marked the debut of the software company Ultimate, Play The Game. Originally written for the Spectrum, Jet Pac set new standards in high-quality graphics on the Sinclair machine and was an instant best-seller. The success of the Spectrum game meant that a version for the Vic-20 followed quickly on its heels.

The 'blast-the-alien' type of arcade-style game is often denigrated for its simplistic, adolescent approach. Nevertheless, the design of a successful example requires considerable programming skill. Jet Pac is such an example.

As is so often the case, the scenario outlined on the cassette inlay seems more complex than it is. As the chief test pilot for the Acme Interstellar Transport Company, it is your job to travel around the galaxy assembling spaceships on assorted planets, while collecting any precious stones and gold that you can lay your hands on. Rocket assembly is greatly assisted by a Hydrovac Jet Pac, which is capable of lifting almost anything, and allows you to manoeuvre components with ease. You are also conveniently armed with Quad Photon Laser Phasers, which are used to blast any alien being that might be unreasonable enough to complain about you despoiling its planet.

The game's description may evoke visions of a glorious Technicolor trip through the varied ecologies of different planets, but as usual the truth is more mundane. The planets you visit are virtually identical, so much so that the test pilot/hero must experience a strong sense of *déjà vu* at each landing. But the aliens make up for this. Each planet is inhabited by a single species only — in fact there would be little room for any others as all the aliens appear to breed like rabbits. These species vary in form from flying saucers to bouncing balloons, but they all have one thing in common — one touch from any one of them means instant death.

Scattered around the surface of each planet are three parts of a spaceship; these must be assembled before you can move on to your next port of call. No screwdrivers or monkey wrenches are required on this job — you simply drop the components onto the rocket base to see the spaceship form before your very eyes.

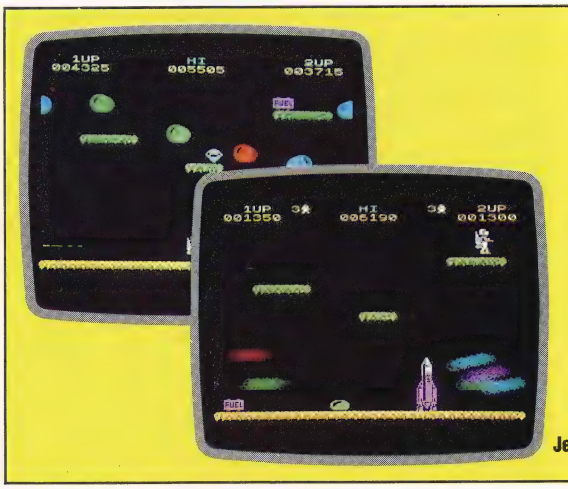
It is the aliens that make this game so enjoyable. There are plenty of them, and at first they give the impression of moving menacingly and unerringly in your direction. After a while you realise that they are, in fact, following predetermined paths from random starting points. The first wave of attackers descends slowly, giving you time to blast them with your deadly laser. The second wave is made up of bouncing balls, which ricochet across the screen between the rock ledges and the ground. This mixture of set paths and random movement provides just the right combination, requiring skill and fast reflexes for successful play.

This type of game is often ruined by a poor choice of control keys. Here they have been chosen sensibly. Two keys on the bottom row of the keyboard are used for left and right movement, catering for both left- and right-handed players. Any key on the second row up fires the lasers, which may be left permanently firing. The row above triggers the jets that make you ascend, and the top row allows you to hover. A nice touch is the fact that if no key is pressed you will drift slowly to the planet surface under the force of gravity.

The graphics are excellent. The test pilot is beautifully drawn, as is the Hydrovac Jet Pac, which gives off convincing puffs of smoke when 'thrust' or 'hover' are activated. The laser streaks the sky with multicoloured lines, and the aliens, when hit, explode in more clouds of smoke.

The game is much the same on either the Spectrum or the Vic-20, although the latter's screen format has the effect of making the test pilot look somewhat overweight. On both machines, Jet Pac is a satisfying and highly addictive game.

Jet Pac: for the 16 or 48K Spectrum, £5.50
for the 8K expanded Vic-20, £5.50
Publishers: Ashby Computers and Graphics Ltd., The Green, Ashby de la Zouch, Leicestershire LE6 5JU
Authors: Ultimate, Play The Game
Joysticks: Kempston Competition-Pro (Spectrum); 'most Commodore-compatible joysticks' (Vic-20)
Format: Cassette



Space Assembly

The Jet Pac test pilot's task of assembling spaceships on various planets is made perilous by the attacking aliens, but the gold and jewels falling from the heavens (!) are some compensation

Jet Pac On The Spectrum

LIZ HEANEY

WINDOW DISPLAY

The ability to create 'windows' on a screen display is a particularly useful facility. We take a detailed look at a machine code program to do just this on the Spectrum, and give an accompanying BASIC demonstration program that shows the effect of scrolling on the screen.

Open Window
The word "HELLO" is scrolled horizontally and vertically around the screen window, one pixel at a time

Our machine code program allows you to define rectangular windows on the Spectrum screen, and to scroll these windows left and right, or up and down. The windows may be of any size and can be placed anywhere on the screen; they do not need to occupy eight by eight pixel character squares.

The machine code program uses tables starting at address $B004 (45060 decimal) to handle window parameters and for temporary data storage. It takes the address of the table for the required window from addresses $B000 and $B001 (45056 and 45057 decimal). The table for each window takes up 11 bytes, so if more than one window is required, the table for the second window will start at $B00F (45071 decimal), the table for the third window will begin at $B01A (45082 decimal), and so on.

The BASIC demonstration program uses one window only. The details of this window are POKEd into memory in lines 180-230, and the window initialisation routine is called at line 240. If more than one window is required, each one must be defined in this way before it can be used. To change windows you must POKE the address of the new window table into memory locations WT and WT+1. Scroll directions are set by POKEing values into memory location WNDWTB + DIR — simply POKE 0 to scroll left, 1 to scroll right, 2 for up, or 3 for down.

The Assembly language program begins by defining a number of constants. PIXADR is a subroutine in the Spectrum ROM that calculates the address of the screen byte, and the bit number within that byte, of a point on the screen that is defined by its PLOT co-ordinates. PIXADR takes the y co-ordinate in the B register and the x co-ordinate in the C register and returns the screen address in the HL register pair and the bit position in the A register.

The routine INITW first checks that the co-ordinates for the bottom right corner of the window are in fact below and to the right of the co-ordinates of the top left corner, and also ensures that the left and right margins are not both in the same byte of screen memory. This last check ensures that the window is at least one character square in width — extra code would be required to scroll a window that is any narrower.

Errors in window initialisation are printed out by the ROM error message routine. The instruction RST 8 (line 2110 of the Assembly listing) calls the ROM routine and returns to BASIC command mode, and the DEFB 25 in line 2120 provides the message 'Q Parameter error'.

The last section of INITW calculates LFTMSK and RTMASK, which are used when scrolling the screen bytes at the window margins, where part of the screen byte may be inside the window and part outside. The individual bits in the masks that correspond to screen bits that lie outside the window margins are set to one, while bits corresponding to bits inside the window area are set to zero.

The scroll program proper begins at the label SCROLL. The program tests the direction of scrolling and calls HORIZ for left or right scrolling, or VERT for up or down scrolling.

Left and right scrolling are very similar in operation, so instead of using two separate routines we have combined them in one. The correct code for each scroll direction is selected by testing bit zero of the direction byte. To see how HORIZ works, we will look at leftward scrolling.

Both left and right scrolling start with the top row of pixels in the window and work downwards, so HORIZ begins by copying the y co-ordinate for the top row into the temporary store for the current row. When scrolling left, we must start at the right-hand end of each row of pixels in the window and work towards the left. To prepare for this, RMASK and LMASK are copied to MASK1 and MASK2 respectively, the address of the screen byte at the left-hand end of the current row of pixels is calculated and stored in the DE register pair, and the address of the screen byte at the right-hand end of the current row of pixels is calculated and stored in the HL register pair. The subroutine HLNSCR is then called to scroll the row of pixels. The routine tests to see if the bottom row of the window has been reached, and if not it takes the next pixel row and jumps back to HORIZ3 to scroll again.

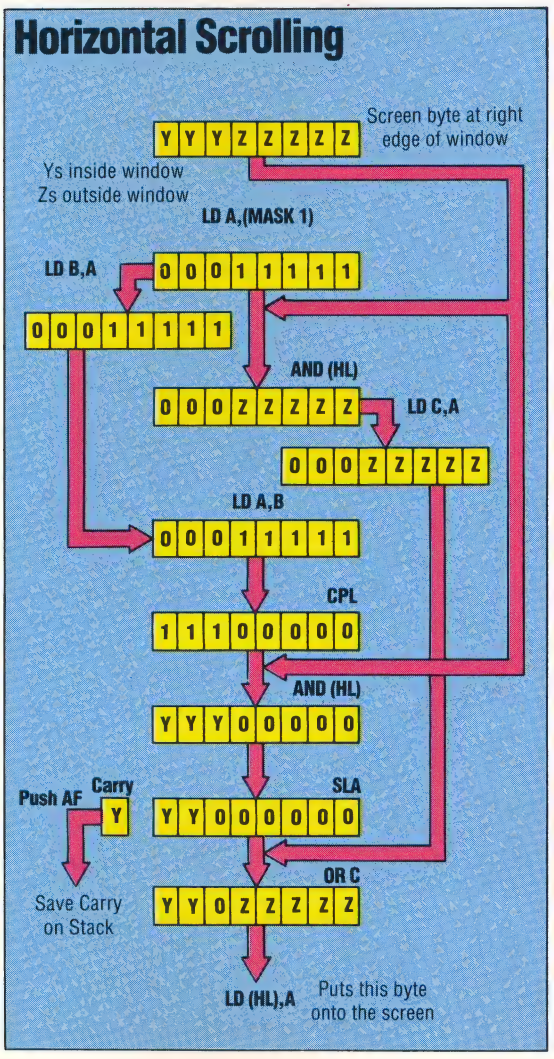
HLNSCR begins at one end of the pixel row, with a byte that may have some bits inside and some outside the window, then moves along the bytes that are contained wholly inside the window and finishes at the other window margin, where again some bits may be inside and some outside the window area. We illustrate this with a diagram that shows how the code operates on the right-hand byte when scrolling left. The section of HLNSCR beginning at NEXT scrolls the bytes inside the window area. The bit that was moved out of the previous byte into the carry flag has been saved on

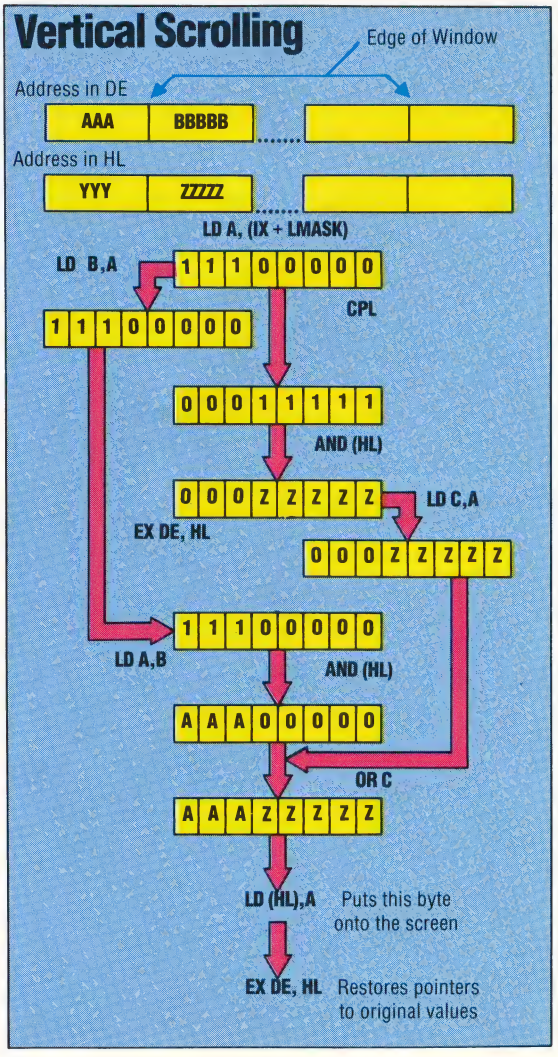
the stack by a PUSH AF instruction, and is restored to the carry flag by POP AF. For a leftward scroll the routine selects the RL (HL) instruction and the byte on the screen is shifted left, with the bit from the carry flag being moved into the left-hand end of the byte and the right-hand bit going into the carry flag. PUSH AF saves the carry flag, so this bit can be moved into the next byte. To test for the end of the row the routine simply compares the L and E registers; this is because the high byte of a screen address will be the same for all screen bytes in the same row. Partial scrolling of the last byte is achieved in a similar way to that of the first byte.

Up and down scrolling routines are combined in a similar way. If we examine what happens when VERT is scrolling upwards, we see that the routine begins by copying the y co-ordinate for the top row into temporary storage for the current row. The screen addresses of both left and right-hand margin bytes in the row are then calculated and the length of the row determined. The routine puts the address of the left-hand margin byte into DE and the address of the corresponding byte of the row immediately below into HL before calling the subroutine VLNSCR to do the scrolling. VERT then tests to see if it has reached the bottom of the window. If it hasn't, it moves down one line before jumping back to VERT5 to scroll another row of pixels. If the bottom has been reached, the section of the routine beginning at CLREDG fills the bottom pixel row with zeros to blank the row on the screen.

VLNSCR treats margin bytes separately, in a similar manner to HLNSCR. The way the code handles these edge bytes is illustrated in our second diagram. To move the central section of the pixel row, the routine increments HL and DE so that they point to the first interior byte on the current line and the corresponding byte on the line above. VLNSCR then calculates the length of the central section (the bytes that fall wholly within the window area), loads this information into BC and uses the block move instruction LDIR to move the whole central section of this row of the window up one line.

These scrolling routines are relatively slow. This is partly because left and right and up and down routines are combined and the program must make frequent tests to decide which piece of code must be used, and partly because the bytes at the left and right-hand margins require special treatment if they straddle the window's borders. The scrolling could be made faster if the program were rewritten to give a separate routine for each direction of scroll, and if the window were restricted to starting and ending at the boundary of a byte of screen memory.





Down The Line

Horizontal scrolling presents particular problems at the window edge: the screen byte must be masked to isolate the pixel bits inside and outside the window, and the byte contents must be shifted. Both processes employ logical AND, OR and SHIFT, and the stack is used to save the PSR status.

Vertical scrolling is made simpler by the Spectrum's screen memory map (see page 358). The screen byte must be masked to isolate pixels inside and outside the window, and the EX instruction is used to swap the contents of the DE and HL registers, which contain screen address pointers

Windows On The Spectrum

Operating Instructions

1) Type in the BASIC Demonstration Program
2) SAVE "SCROLL" LINE 5
3)Type in and RUN the Machine Code Loader program
4) SAVE "SCROLLMC" CODE 45312,410 onto tape directly after the BASIC Program
5) Rewind the tape and LOAD "SCROLL"

Assembly Listing

```
 10 PIXADR EQU  #22AA
 20 WT     EQU  #B000
 30 MASK1  EQU  #B002
 40 MASK2  EQU  #B003
 50 WNDWTB EQU  #B004
 60 LEFTX  EQU  0
 70 TOPY   EQU  1
 80 RIGHTX EQU  2
 90 BOTY   EQU  3
100 LBIT   EQU  4
110 RBIT   EQU  5
120 CURNTY EQU  6
130 DIR    EQU  7
140 LMASK  EQU  8
150 RMASK  EQU  9
160 LENGTH EQU  10
170        ORG  #B100
180 INIT   LD   HL,(WT)
190        PUSH HL
200        POP  IX
210        CALL INITW
220        RET
230 SCROLL LD   HL,(WT)
240        PUSH HL
250        POP  IX
260        BIT  1,(IX+DIR)
270        PUSH AF
280        CALL Z,HORIZ
290        POP  AF
300        CALL NZ,VERT
310        RET
320 HORIZ  LD   A,(IX+TOPY)
330        LD   (IX+CURNTY),A
340        BIT  0,(IX+DIR)
350        JR   Z,HORIZ1
360        LD   B,(IX+LMASK)
370        LD   A,(IX+RMASK)
380        JR   HORIZ2
390 HORIZ1 LD   B,(IX+RMASK)
400        LD   A,(IX+LMASK)
410 HORIZ2 LD   (MASK2),A
420        LD   A,B
430        LD   (MASK1),A
440 HORIZ3 LD   C,(IX+LEFTX)
450        LD   B,(IX+CURNTY)
460        CALL PIXADR
470        EX   DE,HL
480        LD   C,(IX+RIGHTX)
490        LD   B,(IX+CURNTY)
500        CALL PIXADR
510        BIT  0,(IX+DIR)
520        JR   Z,HORIZ4
530        EX   DE,HL
540 HORIZ4 CALL HLNSCR
550        LD   A,(IX+BOTY)
560        CP   (IX+CURNTY)
570        RET  Z
580        DEC  (IX+CURNTY)
590        JR   HORIZ3
600 HLNSCR LD   A,(MASK1)
610        LD   B,A
620        AND  (HL)
630        LD   C,A
640        LD   A,B
650        CPL
660        AND  (HL)
670        BIT  0,(IX+DIR)
680        JR   Z,HLN1
690        SRA  A
700        JR   HLN2
710 HLN1   SLA  A
720 HLN2   PUSH AF
730        OR   C
740        LD   (HL),A
750 NEXT   BIT  0,(IX+DIR)
760        JR   Z,HLN3
770        INC  HL
780        JR   HLN4
790 HLN3   DEC  HL
800 HLN4   LD   A,L
810        CP   E
820        JR   Z,LAST
830        POP  AF
840        BIT  0,(IX+DIR)
850        JR   Z,HLN5
860        RR   (HL)
870        JR   HLN6
880 HLN5   RL   (HL)
890 HLN6   PUSH AF
900        JR   NEXT
910 LAST   LD   C,(HL)
920        LD   A,(MASK2)
930        AND  C
940        LD   B,A
950        POP  AF
960        BIT  0,(IX+DIR)
970        JR   Z,HLN7
980        RR   C
 990        JR   HLN8
1000 HLN7  RL   C
1010 HLN8  LD   A,(MASK2)
1020       CPL
1030       AND  C
1040       OR   B
1050       LD   (HL),A
1060       RET
1070 VERT  LD   C,(IX+LEFTX)
1080       BIT  0,(IX+DIR)
1090       JR   Z,VERT1
1100       LD   B,(IX+BOTY)
1110       JR   VERT2
1120 VERT1 LD   B,(IX+TOPY)
1130 VERT2 LD   (IX+CURNTY),B
1140       CALL PIXADR
1150       PUSH HL
1160       PUSH HL
1170       LD   C,(IX+RIGHTX)
1180       LD   B,(IX+CURNTY)
1190       CALL PIXADR
1200       POP  DE
1210       AND  A
1220       SBC  HL,DE
1230       LD   A,L
1240       DEC  A
1250       LD   (IX+LENGTH),A
1260 VERT3 BIT  0,(IX+DIR)
1270       JR   Z,VERT4
1280       INC  (IX+CURNTY)
1290       JR   VERT5
1300 VERT4 DEC  (IX+CURNTY)
1310 VERT5 LD   C,(IX+LEFTX)
1320       LD   B,(IX+CURNTY)
1330       CALL PIXADR
1340       POP  DE
1350       PUSH HL
1360       CALL VLNSCR
1370       LD   A,(IX+CURNTY)
1380       BIT  0,(IX+DIR)
1390       JR   Z,VERT6
1400       CP   (IX+TOPY)
1410       JR   VERT7
1420 VERT6 CP   (IX+BOTY)
1430 VERT7 JR   NZ,VERT3
1440 CLREDG POP  HL
1450       LD   A,(IX+LMASK)
1460       AND  (HL)
1470       LD   (HL),A
1480       LD   B,(IX+LENGTH)
1490       LD   A,0
1500 CLR1  INC  HL
1510       LD   (HL),A
1520       DJNZ CLR1
1530       INC  HL
1540       LD   A,(IX+RMASK)
1550       AND  (HL)
1560       LD   (HL),A
1570       RET
1580 VLNSCR LD  A,(IX+LMASK)
1590       CALL ENDBYT
1600       INC  HL
1610       INC  DE
1620       LD   B,0
1630       LD   C,(IX+LENGTH)
1640       LDIR
1650       LD   A,(IX+RMASK)
1660 ENDBYT LD  B,A
1670       CPL
1680       AND  (HL)
1690       LD   C,A
1700       EX   DE,HL
1710       LD   A,B
1720       AND  (HL)
1730       OR   C
1740       LD   (HL),A
1750       EX   DE,HL
1760       RET
1770 INITW LD   A,(IX+RIGHTX)
1780       CP   (IX+LEFTX)
1790       JR   Z,ERROR
1800       JR   C,ERROR
1810       LD   A,(IX+TOPY)
1820       CP   (IX+BOTY)
1830       JR   Z,ERROR
1840       JR   C,ERROR
1850       LD   C,(IX+LEFTX)
1860       LD   B,(IX+TOPY)
1870       CALL PIXADR
1880       PUSH HL
1890       LD   (IX+LBIT),A
1900       LD   C,(IX+RIGHTX)
1910       LD   B,(IX+TOPY)
1920       CALL PIXADR
1930       LD   (IX+RBIT),A
1940       POP  BC
1950       LD   A,C
1960       CP   L
1970       JR   Z,ERROR
1980 LFTMSK LD  B,(IX+LBIT)
1990       LD   A,0
2000 L1    SCF
2010       RRA
2020       DJNZ L1
2030       LD   (IX+LMASK),A
2040 RTMASK LD  B,(IX+RBIT)
2050       LD   A,255
2060 L2    AND  A
2070       RRA
2080       DJNZ L2
2090       LD   (IX+RMASK),A
2100       RET
2110 ERROR RST  8
2120       DEFB 25
```

BASIC Demonstration Program

```
  5 CLEAR 32767
 10 LOAD ""CODE
 20 LET WT=45056
 30 LET WINDOWTABLE=45060: REM B004 HEX
 40 LET LEFTX=0
 50 LET TOPY=1
 60 LET RIGHTX=2
 70 LET BOTY=3
 75 LET DIR=7
 80 LET SCROLL=45322
 90 LET INIT=45312
100 BORDER 6
110 PAPER 4: INK 2
120 CLS
180 POKE WT,4: POKE WT+1,176
190 REM WT & WT+1 NOW CONTAIN ADDRESS 45060
IN LO,HI FORMAT
200 POKE WINDOWTABLE+LEFTX,5
210 POKE WINDOWTABLE+TOPY,80
220 POKE WINDOWTABLE+RIGHTX,250
230 POKE WINDOWTABLE+BOTY,35
240 RANDOMIZE USR INIT
250 FOR Y=0 TO 175
260 PLOT 0,Y
270 DRAW 255,0
280 NEXT Y
290 POKE WINDOWTABLE+DIR,2
300 FOR Y=0 TO 45
310 RANDOMIZE USR SCROLL
320 NEXT Y
400 PRINT AT 12,20;"HELLO";
410 POKE WINDOWTABLE+DIR,0
420 FOR I=1 TO 130
430 RANDOMIZE USR SCROLL
440 NEXT I
470 POKE WINDOWTABLE+DIR,3
480 FOR I=1 TO 35
490 RANDOMIZE USR SCROLL
500 NEXT I
510 POKE WINDOWTABLE+DIR,1
520 FOR I=1 TO 130
530 RANDOMIZE USR SCROLL
540 NEXT I
550 POKE WINDOWTABLE+DIR,2
560 FOR I=1 TO 35
570 RANDOMIZE USR SCROLL
580 NEXT I
590 GO TO 410
999 STOP
```

Machine Code Loader

```
100 LET a=45312
110 FOR l=1000 TO 1500 STEP 10
120 LET s=0
130 READ b
140 POKE a,b
150 LET s=s+b
160 NEXT a
170 READ b
180 IF s<>b THEN  PRINT "ERROR IN LINE ";l: STOP
190 NEXT l
1000 DATA 42,0,176,229,221,225,205,69,1167
1010 DATA 178,201,42,0,176,229,221,225,1272
1020 DATA 221,203,7,78,245,204,29,177,1164
1030 DATA 241,196,184,177,201,221,126,1,1347
1040 DATA 221,119,6,221,203,7,70,40,887
1050 DATA 8,221,70,8,221,126,9,24,687
1060 DATA 6,221,70,9,221,126,8,50,711
1070 DATA 3,176,120,50,2,176,221,78,826
1080 DATA 0,221,70,6,205,170,34,235,941
1090 DATA 221,78,2,221,70,6,205,170,973
1100 DATA 34,221,203,7,70,40,1,235,811
1110 DATA 205,103,177,221,126,3,221,190,1246
1120 DATA 6,200,221,53,6,24,215,58,783
1130 DATA 2,176,71,166,79,120,47,166,827
1140 DATA 221,203,7,70,40,4,203,47,795
1150 DATA 24,2,203,39,245,177,119,221,1030
1160 DATA 203,7,70,40,3,35,24,1,383
1170 DATA 43,125,187,40,16,241,221,203,1076
1180 DATA 7,70,40,4,203,30,24,2,380
1190 DATA 203,22,245,24,226,78,58,3,859
1200 DATA 176,161,71,241,221,203,7,70,1150
1210 DATA 40,4,203,25,24,2,203,17,518
1220 DATA 58,3,176,47,161,176,119,201,941
1230 DATA 221,78,0,221,203,7,70,40,840
1240 DATA 5,221,70,3,24,3,221,70,617
1250 DATA 1,221,112,6,205,170,34,229,978
1260 DATA 229,221,78,2,221,70,6,205,1032
1270 DATA 170,34,209,167,237,82,125,61,1085
1280 DATA 221,119,10,221,203,7,70,40,891
1290 DATA 5,221,52,6,24,3,221,53,585
1300 DATA 6,221,78,0,221,70,6,205,807
1310 DATA 170,34,209,229,205,40,178,221,1286
1320 DATA 126,6,221,203,7,70,40,5,678
1330 DATA 221,190,1,24,3,221,190,3,853
1340 DATA 32,209,225,221,126,8,166,119,1106
1350 DATA 221,70,10,62,0,35,119,16,533
1360 DATA 252,35,221,126,9,166,119,201,1129
1370 DATA 221,126,8,205,58,178,35,19,850
1380 DATA 6,0,221,78,10,237,176,221,949
1390 DATA 126,9,71,47,166,79,235,120,853
1400 DATA 166,177,119,235,201,221,126,2,1247
1410 DATA 221,190,0,40,67,56,65,221,860
1420 DATA 126,1,221,190,3,40,57,56,694
1430 DATA 55,221,78,0,221,70,1,205,851
1440 DATA 170,34,229,221,119,4,221,78,1076
1450 DATA 2,221,70,1,205,170,34,221,924
1460 DATA 119,5,193,121,189,40,25,221,913
1470 DATA 70,4,62,0,55,31,16,252,490
1480 DATA 221,119,8,221,70,5,62,255,961
1490 DATA 167,31,16,252,221,119,9,201,1016
1500 DATA 207,25,0,0,0,0,0,0,232
```

SHUFFLE THOSE DIGITS

Our series of short entertaining programs continues with a look at a puzzle known as Reverse. The object of the game is to arrange a list of numbers in ascending order in the least number of moves. This may seem exceedingly simple, but often the best puzzles stem from the simplest concepts.

The program randomly generates a list of numbers for sorting. Changing the order of the numbers is possible only by reversing specified groups within the list. For example, if the computer generates the following random list in response to the player's request for nine numbers:

2 8 4 7 1 5 6 9 3

and the player then specifies 'Reverse? 5', the first five numbers will be inverted and the list becomes:

1 7 4 8 2 5 6 9 3

It shouldn't take you too much time and effort to solve a puzzle like this, and you would expect that it could be easily solved by a predefined algorithm. In practice, however, it is difficult to define a really good one. Suppose that there are n numbers in the list. The most obvious algorithm is this:

- Find the largest number in the list and reverse all the numbers up to its position. The largest number is now at the left-hand end of the list.
- Reverse all n numbers so that the largest number is in its desired position at the right-hand end of the list. This has taken only two reverses.
- Find the second largest number and repeat the whole procedure again. To move this number to its desired location requires a 'Reverse n-1' move.
- Repeat the procedure until the order is obtained.

This algorithm *always* solves the puzzle in 2n-3 moves. But it is possible to achieve a solution in fewer moves than this. To demonstrate how a forward-looking strategy can reduce the number of turns, consider the example we give in the box. Our algorithm would take seven (2×5-3) turns, but a skilled player could do it in four.

This program is a simple example of a whole series of reversing games that people have created and explored. You might like to try your hand at developing games that reverse from either end of the line, or where you have to sort out a grid of numbers rather than just a line. If you do design your own version of the game, you might like to jazz it up by using different coloured blocks to replace the numbers. The object of the game could then be to rearrange a line of blocks to match a pattern of coloured blocks at the top of the screen. You might also like to try incorporating an algorithm into the program to help a player who gets stuck.

Reverse

```
 20 DIM a(20)
 30 CLS : PRINT "Reverse !"
 40 INPUT "How many numbers ? ";n
 50 IF n<0 OR n>20 OR n<>INT n THEN  GO TO 30
 60 REM Jumble the list
 70 FOR i=1 TO n: LET a(i)=i: NEXT i
 80 RANDOMIZE
 90 FOR i=1 TO n
100 LET r=INT (RND*n+1)
110 LET x=a(r): LET a(r)=a(i): LET a(i)=x
120 NEXT i
130 LET t=1
135 REM Print the board
140 CLS : PRINT "Turn ";t;": The list is:": PRINT
150 FOR i=1 TO n: PRINT a(i);" ";: NEXT i
152 REM Check for a win
154 LET i=1
156 IF a(i)=i THEN  LET i=i+1: IF i<=n THEN  GO
TO 156
158 IF i>n THEN  GO TO 230
159 REM Get a go
160 PRINT : PRINT : INPUT "Reverse?";r
170 IF r<>INT r OR r<0 OR r>n THEN  GO TO 140
175 REM Reverse r
180 LET t=t+1
190 FOR i=1 TO INT (r/2)
200 LET x=a(i): LET a(i)=a(r-i+1): LET a(r-i+1)=x
210 NEXT i
220 GO TO 140
230 REM A Winner!
240 PRINT : PRINT : PRINT "You finished in ";t;"
turns"
250 PRINT : INPUT "Play again (y/n) ? ";a$
260 IF a$="Y" OR a$="y" THEN  RUN
270 CLS : STOP
```

Basic Flavours

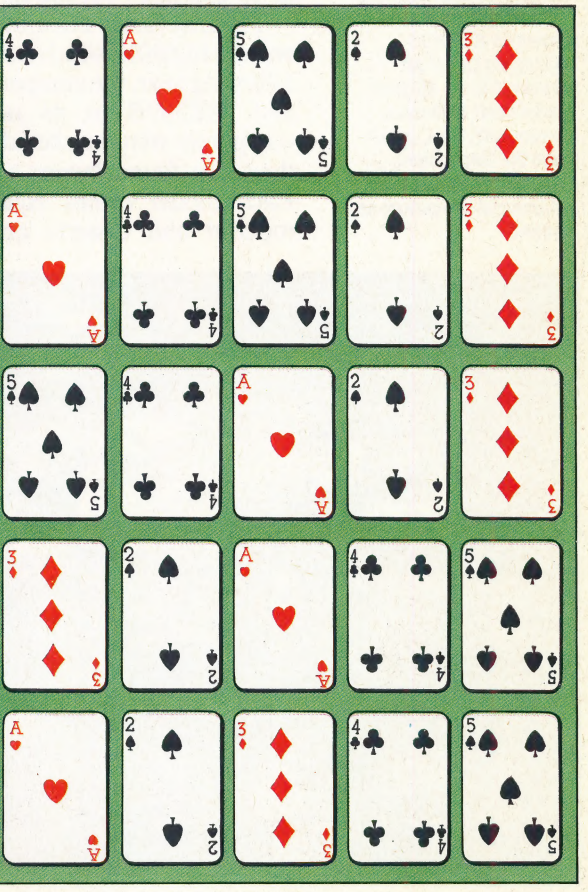
On the Commodore 64 and Vic-20, replace RANDOMIZE by XX=RND(-TI), replace RND*N by RND(1)*N, and replace CLS by PRINT CHR$(147)

On the BBC Micro delete line 80, and replace R=INT(RND*N+1) by R=RND(N)

On the Oric-1 and the Oric Atmos, delete line 80, and replace RND*N by RND(1)*N

About Turn

The object of Reverse is to sort a list by repeatedly reversing subsets of it. The section to be reversed must always start with the leftmost element, and is described by the number of elements included. Here the successful sequence of moves is 2-3-5-3, meaning 'Reverse the leftmost two cards, then the leftmost three, etc.'



GLOBAL ENTERPRISE

The Sharp Corporation is a Japanese-based company that manufactures an enormous range of products, from transistors and refrigerators to computers and industrial robots. International sales in 1983, the year it entered the UK home computer market, amounted to nearly £2 billion.

The Sharp Corporation has always been party to major technological innovation. But its first successful product was an extremely humble item — the 'Ever Sharp' propelling pencil. Its creator, Tokuji Hayakawa, set up Sharp in 1915 to manufacture his invention; and the company expanded steadily in succeeding years. In 1925 it moved into electronics with a crystal radio set; and entry into the world's consumer electronics markets came in the post-war years when it began producing television sets and other domestic appliances. In the mid-1960s the company intervened in the business machinery market with a series of desk-top calculators. Today it is a huge multinational corporation, subdivided into six manufacturing groups, with 34 production plants in 30 countries outside Japan.

The first Sharp computer marketed in the UK was the MZ80K, which was launched in 1981. The following year, the company added the MZ380A and MZ380B to its range. Although these computers were marketed as business machines, they also found favour with home micro users. Each model comes equipped with a built-in monitor and cassette drives. Originally, these computers were marketed as 'clean machines', emphasising their lack of a resident language in ROM. The advantage of this was that a variety of languages, including CP/M, could be loaded on-board from cassette.

Sharp began selling a full range of business and home computers in the UK at the beginning of 1983, when the company's list of products was extended to include the MZ-3541 business machine and the PC-1500 pocket computer. The success of the latter led to the marketing of the PC-1251 pocket computer.

The company entered the home computer market with the launch of the Sharp MZ-711. This is the European version of the Japanese MZ-700 series, and the enormous Japanese character set of the original has allowed room for extra graphics facilities on the European model. The machine has a data recorder fitted as standard, and space is included for an optional printer/plotter.

In May 1984, the company released the PC-1500A, an upgrade of the PC-1500. The new model is fitted with 8.9 Kbytes of RAM, which can be expanded up to 24 Kbytes. In the autumn of 1984 the company plans to launch the PC-1350 pocket machine with a four-line display and graphics capability. Sharp Corporation also intends to introduce the Sharpwriter package, which is a marriage of the Sharp ZX-401 electric typewriter and the MZ-3541 microcomputer. The typewriter is used as a keyboard/printer connected to an RS232 interface in the computer.

Asked about further developments from Sharp, sales director Rod Goodier says that he is 'personally very keen to expand the pocket computer market, where there is very great potential'. This does not mean that the home computer market will be neglected. 'With the MZ-700 family we will really go to town, and we intend to stay firmly in the home computer market.' Company spokesman Peter Fletcher explains: 'The business equipment division, including home computers, is a relatively recent innovation, and represents 25 per cent of turnover in the UK. The plan is to increase the proportion of turnover.'

At the moment Sharp UK has only a warehousing and marketing operation. The company is currently building a £15 million plant in Wrexham to assemble video recorders. The plant is expected to produce 60,000 machines in 1985 for distribution throughout Europe.

On the question of Sharp's involvement with other Japanese companies in the expected MSX invasion in the autumn, Rod Goodier replies that 'Sharp have developed an MSX system, but there are no plans to launch in the UK at the moment.'

Industrial Instruments Group
Technological subjects researched at Sharp's Engineering Centre are transferred to the IIG plant (shown here) at Yamato-Koriyama-shi, Nara, Japan, where the Corporation's products (including calculators and personal computers) are designed

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

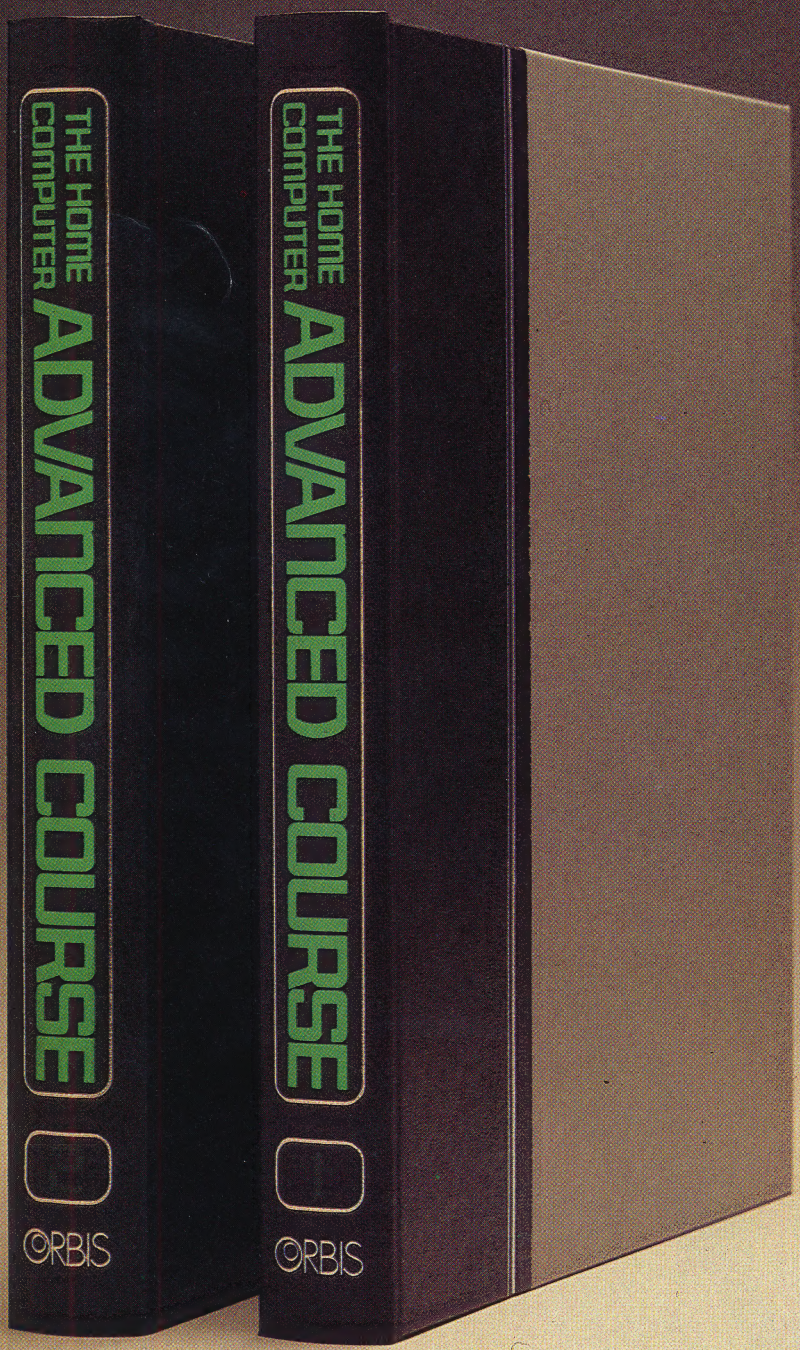
No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.



All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.

Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.

The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.